Integration des intelligenten tutoriellen Systems eTutor in die Lernumgebung Moodle

Diplomarbeit

Zur Erlangung des akademischen Grades
„Magister der Sozial- und Wirtschaftswissenschaften"
(Mag.rec.soc.oec.)
im Diplomstudium Wirtschaftsinformatik

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik -
Data & Knowledge Engineering

Betreuer: o. Univ.-Prof. Dr. Michael Schrefl
Mitbetreuender Assistent: Mag. Christian Eichinger

Verfasst von: Christian Fischer

Linz, im Juni 2010

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Diplomarbeit wurde von mir weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt.

Linz, ____________________ ____________________

Ort, Datum Unterschrift

Kurzfassung

Unter dem Begriff *E-Learning* wird der Einsatz von Computersystemen zur Unterstützung von Lernprozessen verstanden. Die dabei verwendeten Lernsysteme werden in zwei Kategorien eingeteilt: *Learning Management Systeme* unterstützen die Verwaltung von Lernmaterialien in Kursen während sich *intelligente tutorielle Systeme* auf die automatische Korrektur umfangreicher Aufgabenstellungen konzentrieren. Im Zuge dieser Diplomarbeit sollen mittels einer Integration des intelligenten tutoriellen Systems *eTutor* in das frei verfügbare Learning Management System *Moodle* die Vorzüge beider Lernsystem-Typen vereint werden.

Bei der Umstellung auf ein integriertes Lernsystem mussten einige Rahmenbedingungen eingehalten werden. An der Johannes Kepler Universität wird Moodle fakultätsübergreifend genutzt und zentral verwaltet. Änderungen am Moodle-Kernsystem sollten vermieden werden, da diese im Zuge einer Aktualisierung der Lernumgebung auf neue Moodle-Versionen migriert werden müssten. Außerdem sollte es möglich sein, bestehende Übungsbeispiele und die dafür erforderliche Korrekturfunktionalität des am Institut für Wirtschaftsinformatik – Data and Knowledge Engineering entwickelten eTutor-Systems mit möglichst wenig Umstellungsaufwand weiter zu verwenden.

Zunächst wurde eine geeignete Vorgehensweise zur zielgerichteten Migration in ein integriertes Lernsystem ausgearbeitet, die daraufhin umgesetzt wurde. Im resultierenden Lernsystem sind die beteiligten Komponenten lose gekoppelt, wodurch der Änderungsaufwand für Bestandteile des Systems minimal gehalten wird. So genügt eine Anpassung von Konfigurationsdateien, um neue Module zur Erweiterung der Korrekturfunktionalität in das Lernsystem einzubinden.

Durch die Integration von Moodle und eTutor ergeben sich Vorteile sowohl für Kursteilnehmer als auch für Lernsystem-Administratoren. Studenten müssen sich gemäß dem Prinzip des Single Sign-on nur noch auf einem Lernsystem einloggen, da Kursdaten zentral im Moodle-System administriert werden. Darüber hinaus wurde eine Unterstützung mehrsprachiger Benutzerdialoge implementiert, welche die Übungsausarbeitung speziell für fremdsprachige Austauschstudenten erleichtert.

Abstract

E-learning is defined as the usage of computer systems to support learning processes. The learning systems that are used for this purpose are classified into two categories: *Learning Management Systems* focus on functionality for the administration of instruction material during courses. *Intelligent Tutoring Systems* concentrate on the automatic correction of comprehensive task assignments. The goal of this diploma thesis is to combine the advantages of both types of e-learning systems by integrating the Intelligent Tutoring System *eTutor* into the Learning Management System *Moodle*.

The integration had to be performed considering multiple constraints. At the Johannes Kepler University, Moodle is used for several faculties and it is administrated centrally. Therefore, the Moodle core system should not be modified, as changes therein would require regular adaptions to subsequently Moodle versions. Whenever a new Moodle release is installed, implemented changes would get lost otherwise. The learning system eTutor was developed at the Department of Business Informatics - Data and Knowledge Engineering and has been used successfully in the courses for a number of years. The goal is to reuse existing exercises and expert correction functionality in order to minimise time and effort of the system migration.

The components of the integrated Moodle-eTutor system are loosely coupled. Consequently, the effort to modify parts of the system is minimised. For example, an adaptation of configuration files is sufficient, in order to include new expert correction modules into the learning system. Course participants and administrators benefit from the integration of Moodle and eTutor. Students just need to log-in to one learning system (single sign-on) because the course data is administrated centrally in the Moodle system. Furthermore the support of multilingual user dialogues allows exchange students who only speak foreign languages to participate in Moodle eTutor courses.

Inhaltsverzeichnis

Abkürzungen

AOP	Aspect-oriented programming
CORBA	Common Object Request Broker Architecture
CSCW	Computer Supported Cooperative Work
DAO	Data Access Object
DI	Dependency Injection
DKE	Institut für Wirtschaftsinformatik - Data and Knowledge Engineering
FIFO	First In First Out
GPL	GNU General Public License
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IoC	Inversion of Control
IS	Informationssystem
ITS	Intelligent Tutoring System (deutsch: intelligentes tutorielles System)
JDBC	Java Database Connectivity
JKU	Johannes Kepler Universität
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
LMS	Learning Management System
MVC	Model View Controller
ORB	Object Request Broker
ORM	Object-Relational Mapping
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
POJO	Plain Old Java Object
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UML	Unified Modelling Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
XQuery	XML Query Language
XSLT	Extensible Stylesheet Language Transformation

Abbildungsverzeichnis

Codeausschnittverzeichnis

1 Aufgabenstellung

Der Einsatz von Computertechnologie ermöglicht kostengünstiges und bedarfsgerechtes Lernen. Im Gegensatz zu Lehrern im klassischen Unterricht, können Lernsysteme zur Unterstützung des Lernprozesses örtlich und zeitlich flexibel verwendet werden (Zhang u. a. 2004, S.76). Es ist somit möglich, Lehrende von Routineaufgaben zu entlasten, Kursteilnehmer abhängig von ihrem Lernfortschritt individuell zu betreuen, und Übungsaufgaben automatisch vom Lernsystem korrigieren zu lassen (Lusti 1992).

Der Einsatz von Computern als Lehrmedium, mit der Berücksichtigung technischer und didaktischer Erkenntnisse, wird als E-Learning bezeichnet (Zhang u. a. 2004, S.76). E-Learning-Systeme werden sowohl in Bildungseinrichtungen als auch in Unternehmen eingesetzt. An der Johannes Kepler Universität (JKU) kann zum Beispiel das *Multimediastudium Rechtswissenschaften* vollkommen ohne physische Anwesenheit an der Universität absolviert werden. Das Unternehmen IBM nutzt E-Learning, um Mitarbeiter im Verkauf, die an verschiedenen Standorten tätig sind, über neue Produkte zu informieren (Arnold 2007).

E-Learning-Systeme können unter anderem in *Learning Management Systems* (LMS) und *Intelligent Tutoring Systems* (ITS – auf Deutsch: intelligente tutorielle Systeme) eingeteilt werden. Learning Management Systems dienen zur Verwaltung von Kursmaterialien und Kursteilnehmern in E-Learning-Kursen. LMS werden daher meist unterstützend zu bestehenden Lernangeboten eingesetzt. Neben kommerziellen Lösungen wie der Blackboard Learn Platform (Blackboard 2010) existieren frei verfügbare LMS wie Moodle (Moodle 2010) oder Sakai (Sakai 2010), die unter einer Open Source-Lizenz verfügbar sind und von einer Community weiterentwickelt werden.

Demgegenüber werden unter dem Begriff *Intelligent Tutoring Systems* wissensbasierte Lernsysteme verstanden, die das Ziel verfolgen, fachliche und pädagogische Kompetenz eines idealen menschlichen Lehrers am Computer nachzubilden (Lusti 1992). ITS sollen vor allem die Problemlösungsfähigkeiten des Lernenden verbessern (Brooks u. a. 2006). Aus diesem Grund ermöglichen derartige E-Learning-Systeme die Bearbeitung komplexer Übungsaufgaben. Diese werden durch sogenannte Expertenmodule unterstützt, welche die Korrekturfunktionalität zu einem Themengebiet abbilden. Idealerweise sind ITS in der Lage, alternative Lösungen des Lerners zu beurteilen, und bieten Erklärungsschritte und Hilfen an.

1.1 Ausgangssituation

Das Institut für Wirtschaftsinformatik - Data and Knowledge Engineering an der JKU entwickelte zur Unterstützung der Lehrveranstaltungen *Übung Datenmodellierung* und *Übung Data & Knowledge Engineering* das webbasierte E-Learning-System *eTutor* (Hofer 2005). Es verwaltet Kursteilnehmer samt deren Bewertungen und wird zur automatisierten Korrektur von Hausübungen eingesetzt. Für eine automatisierte Beispielausarbeitung eignen sich Beispieltypen, bei denen eine algorithmische

Problemlösung möglich ist. Als Beispiel kann das Decompose-Verfahren zum Normalisieren von Relationen genannt werden. eTutor unterstützt auch die manuelle Übungskorrektur durch studentische Mitarbeiter im Lehrbetrieb (im Folgenden als Tutoren bezeichnet). Tutoren kontrollieren dabei jene Übungstypen, für die im eTutor kein Expertenmodul zur Aufgabenkorrektur implementiert wurde.

Kernfunktionalität des eTutor-Systems ist die automatische Korrektur von Übungsbeispielen. Da zum Zeitpunkt der eTutor-Erstellung keine geeigneten LMS vorhanden waren, musste die Verwaltungsfunktionalität für Kurse, Teilnehmer und Bewertungen vom Entwicklerteam am DKE-Institut selbst programmiert werden. Diese Funktionalität war notwendig, um Korrekturergebnisse Studenten zuordnen zu können. Da bei der Erstellung des eTutors die Entwicklung von Expertenmodulen im Vordergrund stand, wurde sie jedoch nur rudimentär umgesetzt. Daraus ergibt sich eine Reihe von Problemen, wie eine fehlende Unterstützung von Mehrsprachigkeit in der Benutzerschnittstelle. Weitere Verbesserungspotentiale des derzeitigen Systems werden in Kapitel 4.2.1.4 - Zielsystem-Funktionalität beschrieben.

1.2 Zielsetzung

Das eTutor-System des DKE-Instituts unterstützt die automatische Übungskorrektur komplexer Beispieltypen und kann daher als intelligentes tutorielles System (ITS) eingestuft werden. Darüber hinaus werden Aufgaben eines Learning Management System, wie die Verwaltung von Kursdaten, übernommen. Um die Probleme des eTutors in seiner Verwaltungsfunktionalität zu beheben, sollen im Zuge der Diplomarbeit die Expertenmodule des eTutor-Systems in ein vorhandenes Learning Management System (LMS) integriert werden. Ein Vergleich verschiedener LMS ist nicht erforderlich, da der Einsatz des LMS *Moodle* vorgegeben ist. Das Moodle-System wird an der JKU zentral angeboten und an allen Fakultäten der Universität zur Kursunterstützung eingesetzt, weshalb der eTutor in das LMS *Moodle* integriert werden soll. Trotzdem ist in der Umsetzung auf Portabilität zu anderen LMS zu achten.

Den Umgang mit Moodle sind sowohl Kursteilnehmer als auch Kursleiter gewohnt. Ein weiterer Vorteil dieser eTutor-Integration besteht darin, dass der vorhandene Moodle-Login, der jedem Student an der JKU zugewiesen wird, auch für den eTutor verwendet werden kann und eine gesonderte Registrierung am eTutor entfällt. Für die Kursadministration und die Verwaltung der Studenten soll daher Moodle eingesetzt werden. eTutor übernimmt ausschließlich die automatische Korrektur von Übungsbeispielen.

Für die Integration des ITS eTutor in das LMS Moodle werden folgende Ziele definiert:

- Die bestehende Funktionalität des eTutor-Expertensystems soll weitgehend beibehalten werden, weshalb vorhandene Expertenmodule mit der Lernumgebung lose gekoppelt werden sollen.

- Es sollen mehrsprachige Übungsaufgaben unterstützt werden.
- Bestehende Daten im eTutor-System sollen bei Bedarf übernommen werden können.
- Änderungen am Moodle-Kernsystem sollen vermieden werden.

Rahmenbedingungen wie die organisatorische Gestaltung der Lehrveranstaltungen oder der Einsatz von Moodle als fakultätsübergreifendes LMS sind vorgegeben. Gegenstand der Diplomarbeit ist daher nicht die Abstimmung des computergestützten Lernens mit dem herkömmlichen Lernen (beispielsweise physische Anwesenheit bei sogenannten Präsenzphasen). Derartige Aufgaben sind Teil des Fachgebiets Blended Learning (Garrison & Kanuka 2004). Zudem bestehen Einschränkungen bei der Gestaltung der Integration. So dürfen am Moodle-Kernsystem keine Änderungen durchgeführt werden, da ein Versionsupdate von Moodle ohne Anpassung des eTutor-Integrationsmoduls durchführbar bleiben muss.

In Kapitel 2 ist dargestellt, wie ein ideales ITS in der Literatur beschrieben wird. Der vorgestellte modulare Ansatz soll für das eTutor-System übernommen werden. Dazu wird das derzeitige eTutor-System, auch als Altsystem bezeichnet, in eine Moodle-Umgebung, das sogenannte Zielsystem, übergeführt. Dieser als Migration bezeichnete Umstellungsvorgang wird in Kapitel 3 allgemein beschrieben, während Kapitel 4 die konkrete eTutor-Migration behandelt. In Kapitel 5 wird die Diplomarbeit zusammengefasst und die erstellte Lösung beurteilt.

2 Aufbau eines intelligenten tutoriellen Systems

Ziel von intelligenten tutoriellen Systemen (ITS) ist es, dem Lernfortschritt angepasste Lerninhalte bereitzustellen (Brusilovsky 2003, S.2). Skripten dienen dazu, Wissenslücken zu schließen, und Übungsbeispiele sollen Defizite ausmerzen. Durch die gezielte Auswahl von Skripten und Übungsbeispielen kann der Lernprozess an die individuellen Bedürfnisse der Studierenden angepasst werden.

Learning Management Systems (LMS) eignen sich vor allem für das Bereitstellen und Verwalten von Skripten. Der Schwerpunkt von ITS liegt auf der Bearbeitung komplexer Übungsbeispiele, wobei Übungsabgaben automatisiert ausgewertet werden können. Sie bieten Hilfestellungen, die sich schrittweise dem Wissensstand des Studenten angleichen. Dazu werden vom ITS dynamisch Rückmeldungen erstellt, die nicht vom Übungsautor vorbereitet wurden. Um derart komplexe Aufgaben erfüllen zu können, sind die Komponenten eines ITS meist eng miteinander verzahnt (Brusilovsky 2003).

Im Folgenden wird dargestellt, wie ein idealtypisches ITS in der Literatur beschrieben wird (Lusti 1992, S.26). Das ITS wird dazu, wie in Abbildung 1 ersichtlich, in vier Module aufgeteilt, wobei ein Modul als ein möglichst selbständiger Systembaustein definiert ist.

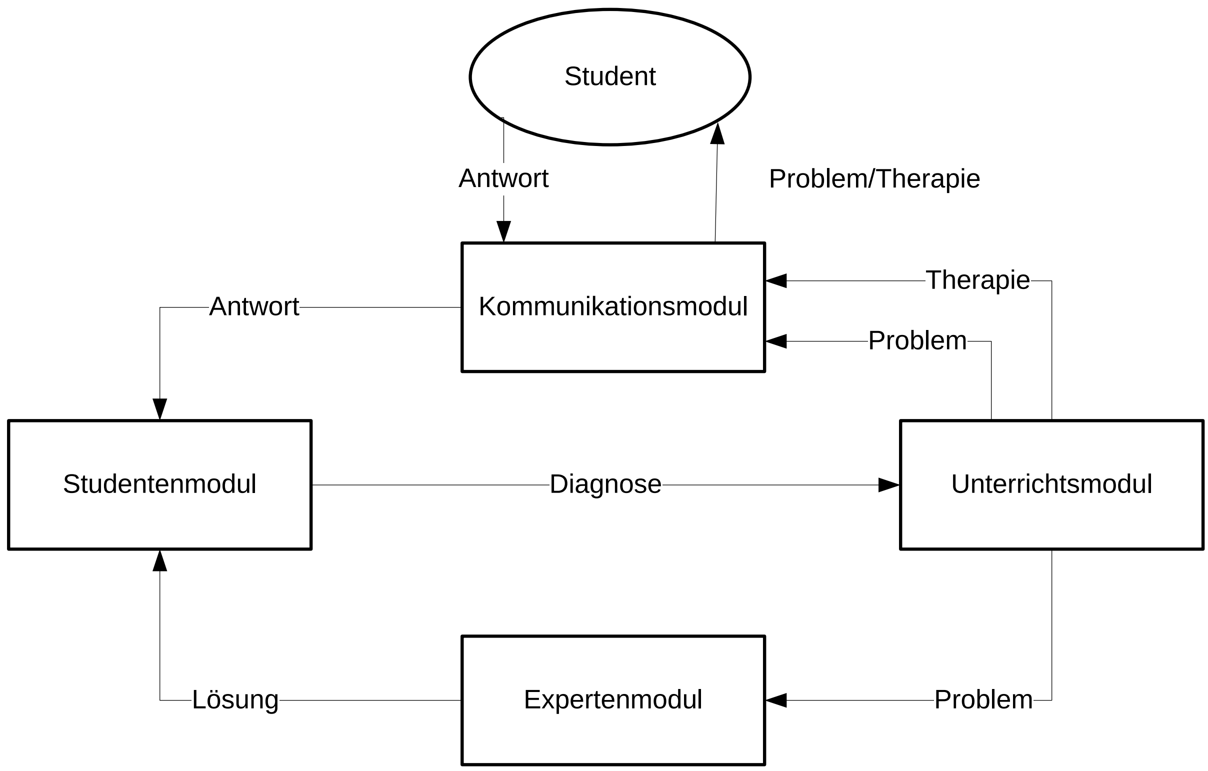


Abbildung 1 - idealtypisches ITS in Anlehnung an (Lusti 1992)

Das *Expertenmodul* versucht, die Fachkompetenz eines Lehrers nachzubilden. Es versetzt ein idealtypisches ITS in die Lage, Ausarbeitungen automatisch zu korrigieren. Mit der vom

Expertenmodul bereitgestellten Musterlösung ist das Studentenmodul in der Lage, eine dem Lernenden angepasste Diagnose zurückzuliefern, die vom Unterrichtsmodul und vom Kommunikationsmodul aufbereitet und angezeigt wird. Die gefundenen Fehler werden schrittweise erklärt, damit der Student im Lernprozess Fortschritte machen kann. Aus diesem Grund ähneln Expertenmodule Expertensystemen (Lusti 1992, S.17), die ebenfalls aus einer Problemlösungs- und einer Erklärungskomponente bestehen. Expertenmodule können Aufgabengeneratoren enthalten, die Aufgabenstellungen mittels Parameter an den Kenntnisstand des aktuellen Benutzers anpassen. Angaben komplexerer Übungsaufgaben, beispielsweise Textaufgaben von Fallstudien, sind hingegen vorgegeben und werden in einer Aufgabenbank verwaltet.

Im *Studentenmodul* wird der aktuelle Lernfortschritt des Studenten festgehalten. Neben dem Fähigkeitsstand wird auch der Motivationsstand erfasst. Löst ein Student ein Beispiel, versucht das Studentenmodul, systematische Fehler des Lernenden festzustellen und daraufhin eine Fehlerdiagnose zu geben.

Die in Abbildung 1 vom Expertenmodul ausgearbeitete Diagnose beschreibt die Schwächen eines Studenten und wird vom *Unterrichtsmodul* dazu verwendet, eine sogenannte Therapie vorzuschlagen. Dabei handelt es sich entweder um weitere Übungsbeispiele oder Lernskripten, die vom Studenten bearbeitet werden sollen, um seine Fehlerwahrscheinlichkeit zu reduzieren. Das Unterrichtsmodul legt somit fest, welche Kursinhalte (Aufgaben und Informationen) dem Studenten in welcher Reihenfolge angeboten werden.

Sobald die Interaktionsinhalte festgelegt sind, wird vom *Kommunikationsmodul* eine adäquate Interaktionsform gewählt. Die zu vermittelnden Inhalte sind in einem idealtypischen ITS rhetorisch und pädagogisch wirksam aufbereitet.

Zum Themengebiet *Data and Knowledge Engineering* existiert bereits eine Reihe von ITS (Faeskorn-Woyke u. a. 2009), von denen jedoch kein System alle Kriterien eines idealen ITS erfüllt. Für eine detaillierte Funktionsübersicht dieser Systeme wird auf die Diplomarbeit von Anita Hofer verwiesen (Hofer 2005, S.52), im Zuge derer das eTutor-Altsystem entwickelt wurde.

3 Migrationstechniken zur Systemintegration

Ziel des Migrationsprozesses ist es, Software und Hardware eines unzureichenden und schlecht zu wartenden Altsystems in ein geeignetes und robustes Zielsystem überzuführen, ohne dabei den Systembetrieb zu beeinträchtigen. Ein Zielsystem zeichnet sich dadurch aus, dass es aktuellen Ansprüchen genügt und aufgrund seiner flexiblen Gestaltung schnell und ohne großen Aufwand an zukünftige Anforderungen angepasst werden kann. Es kann noch Teile des Altsystems beinhalten und muss daher nicht von Grund auf neu entwickelt werden (Bisbal u. a. 1999, S.3). Ein Altsystem ist wie folgt definiert:

„A legacy information system is any information system that significantly resists modification and evolution to meet new and constantly changing business requirements.” (Brodie & Stonebraker 1995, S.3)

Als Altsysteme werden daher geschäftskritische Informationssysteme bezeichnet, die, laut Stonebrakers Definition, aktuellen Anforderungen nicht mehr entsprechen und sich nur mühsam anpassen lassen. Sie laufen meist in einer unzeitgemäßen Systemumgebung und sind nur mit großem Aufwand instand zu halten. Ein Grund für eine aufwendige Erweiterbarkeit kann mangelhafte Dokumentation sein. Zudem fehlen oft klare Schnittstellen zu anderen Informationssystemen, weshalb sich die Einarbeitung in den Systemaufbau meist als kosten- und zeitintensiv erweist. Es ist somit schwierig, aber nicht unmöglich, Altsysteme zu erweitern.

In dieser Arbeit werden Migrationstechniken nur unter dem Gesichtspunkt der Integration der Lernsysteme *Moodle* und *eTutor* betrachtet. Kontinuierliche und für den laufenden Betrieb notwendige Wartung eines Altsystems wird im Rahmen dieser Arbeit ausgeklammert. Diese Problemstellung wird beispielsweise von Schneidewind (Schneidewind 1998) erörtert.

3.1 Aufbau eines Informationssystems

Ein Informationssystem besteht aus Funktionen, Daten und Schnittstellen, wobei letztere entweder Benutzerschnittstellen oder Systemschnittstellen zugeordnet werden können. Informationssysteme (IS) können je nach Kapselung ihrer Bestandteile klassifiziert werden in (Brodie & Stonebraker 1995): *zerlegbare IS* (siehe Abbildung 2), *teilweise zerlegbare IS* (Abbildung 3) und *nicht zerlegbare IS* (Abbildung 4). *Zerlegbare Informationssysteme* haben die Eigenschaft, dass Schnittstellen (Systemschnittstellen und Benutzeroberflächen), Anwendungsmodule und Datenbankzugriff voneinander abgegrenzt sind. Beispielsweise können Daten, die in den Tabellen einer relationalen Datenbank abgelegt sind, besser vom Rest des Informationssystems abgegrenzt werden als der Inhalt einer objektorientierten Datenbank, weil die darin gespeicherten Objekte neben Daten auch Funktionen bereitstellen können. Im Zuge der Diplomarbeit werden ausschließlich Systeme mit relationalen Datenbanken behandelt. Aus diesem Grund ist die Datenbanksicht für die Migration von großer Bedeutung.

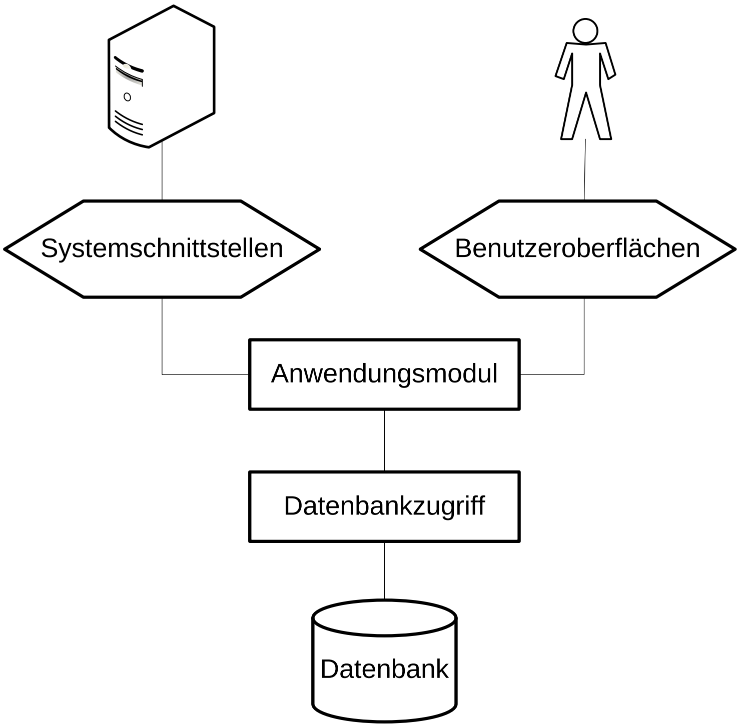


Abbildung 2 - zerlegbares IS in Anlehnung an (Brodie & Stonebraker 1995, S.16)

Bei *teilweise zerlegbaren Informationssystemen* (Abbildung 3) kann das Anwendungsmodul nicht vom Datenbankzugriff isoliert betrachtet werden. Das ist dann der Fall, wenn technische Kapselungsmöglichkeiten nicht ausgeschöpft werden, also beispielsweise direkt im Anwendungscode SQL-Statements abgesetzt werden und auf eine Abstraktionsschicht für den Datenzugriff verzichtet wird. Eine mangelhafte Strukturiertheit erschwert die Analyse des Altsystems und resultiert in fehleranfälligeren Umstellungen.

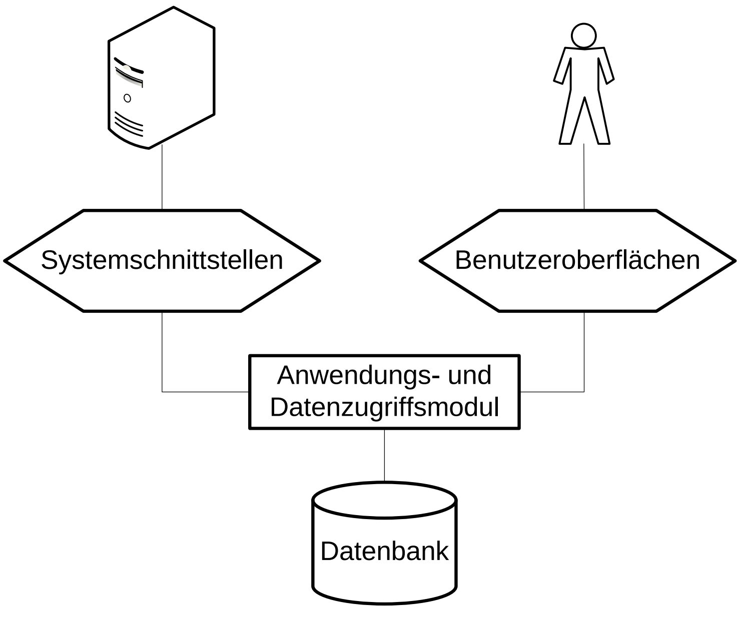


Abbildung 3 - teilweise zerlegbares IS in Anlehnung an (Brodie & Stonebraker 1995, S.17)

Ein *nicht zerlegbaren Informationssystem* (Abbildung 4) ist dadurch charakterisiert, dass das System in keine separablen Komponenten aufgeteilt werden kann. Aufgrund der engen Verzahnung muss das Altsystem als eine Black Box angesehen werden, wodurch eine schrittweise Migration erschwert wird.

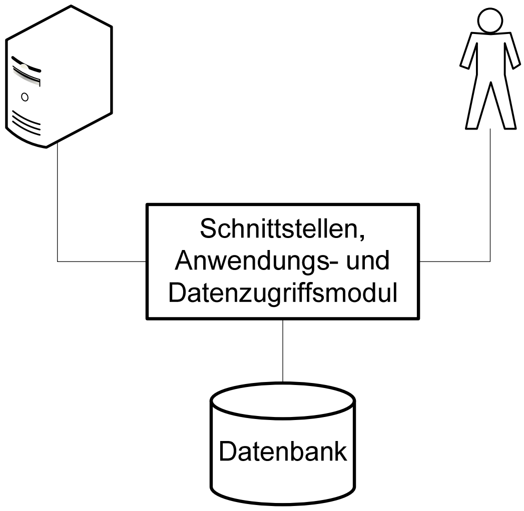


Abbildung 4 - nicht zerlegbares IS in Anlehnung an (Brodie & Stonebraker 1995, S.18)

3.2 Vorgehensweisen der Migration

Um von einem Alt- in ein Zielsystem zu migrieren existieren unterschiedliche Ansätze, von denen in diesem Abschnitt fünf Vorgehensweisen beschrieben werden.

3.2.1 Umstellung auf einen Schlag (Cold Turkey)

Bei der Migrationsart *Cold Turkey* wird das laufende System zunächst abgeschaltet und daraufhin werden Funktionen und Daten des Altsystems durch ein völlig neu entwickeltes Zielsystem ersetzt (Brodie & Stonebraker 1995). Dieser „cutover“ (Umstellungsschritt) wird auch als Big Bang bezeichnet (Bisbal u. a. 1999, S.4). Da die Neuentwicklung vor der Umstellung noch nicht unter Realbedingungen getestet wurde, ist das Risiko zu scheitern, speziell bei großen Projekten, hoch. Ein Umstellungsmisserfolg wegen fehlender Funktionalität kann beispielsweise aus mangelhaft erhobenen Anforderungen resultieren. Zudem kann nicht sichergestellt werden, dass sich während der Entwicklung des Zielsystems keine Geschäftsprozesse ändern und sich somit neue Anforderungen ergeben. Läuft die Umstellung jedoch problemlos, steht die gesamte Funktionalität des Zielsystems sofort nach der Umstellungsphase zur Verfügung.

3.2.2 Schrittweise Umstellung (Chicken Little)

Bei einer *Chicken Little*-Migration wird iterativ vorgegangen. Vom Altsystem werden schrittweise Funktionen und Daten auf das Zielsystem umgestellt, wobei jeder dieser Schritte das System von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt und dadurch das System dem definierten Ziel näherbringt (Brodie & Stonebraker 1995). Der Umfang des Zielsystems wächst daher kontinuierlich an. Je weniger Änderungen mit einer Migrationsphase durchgeführt werden,

desto geringer ist die Anzahl der Verbesserungen und desto überschaubarer ist das Risiko. Schlägt eine Phase fehl, kann auf die vorherige Umstellungsstufe zurückgewechselt werden – vorangehende Schritte bleiben erhalten. Im Gegensatz zur Cold Turkey-Strategie gefährden daher einzelne Probleme bei der Umstellung nicht das gesamte Projekt sondern nur die aktuelle Iteration. Zudem ist aufgrund der Teilerfolge in den einzelnen Umstellungsphasen der Projektabschluss besser planbar als bei einer Big Bang-Umstellung. Bei letzterer kann eine Migration erst nach dem Umschalten auf das Zielsystem als Gesamterfolg oder als Misserfolg klassifiziert werden.

In Abbildung 5 sind die Schritte einer Iterationsphase der Chicken Little-Migration angeführt. Abhängigkeiten zwischen den Migrationsschritten werden mittels Pfeilen dargestellt. Solange keine Abhängigkeiten verletzt werden, erlaubt es der Chicken Litte-Ansatz, Schritte zu vertauschen oder zusammenzulegen.

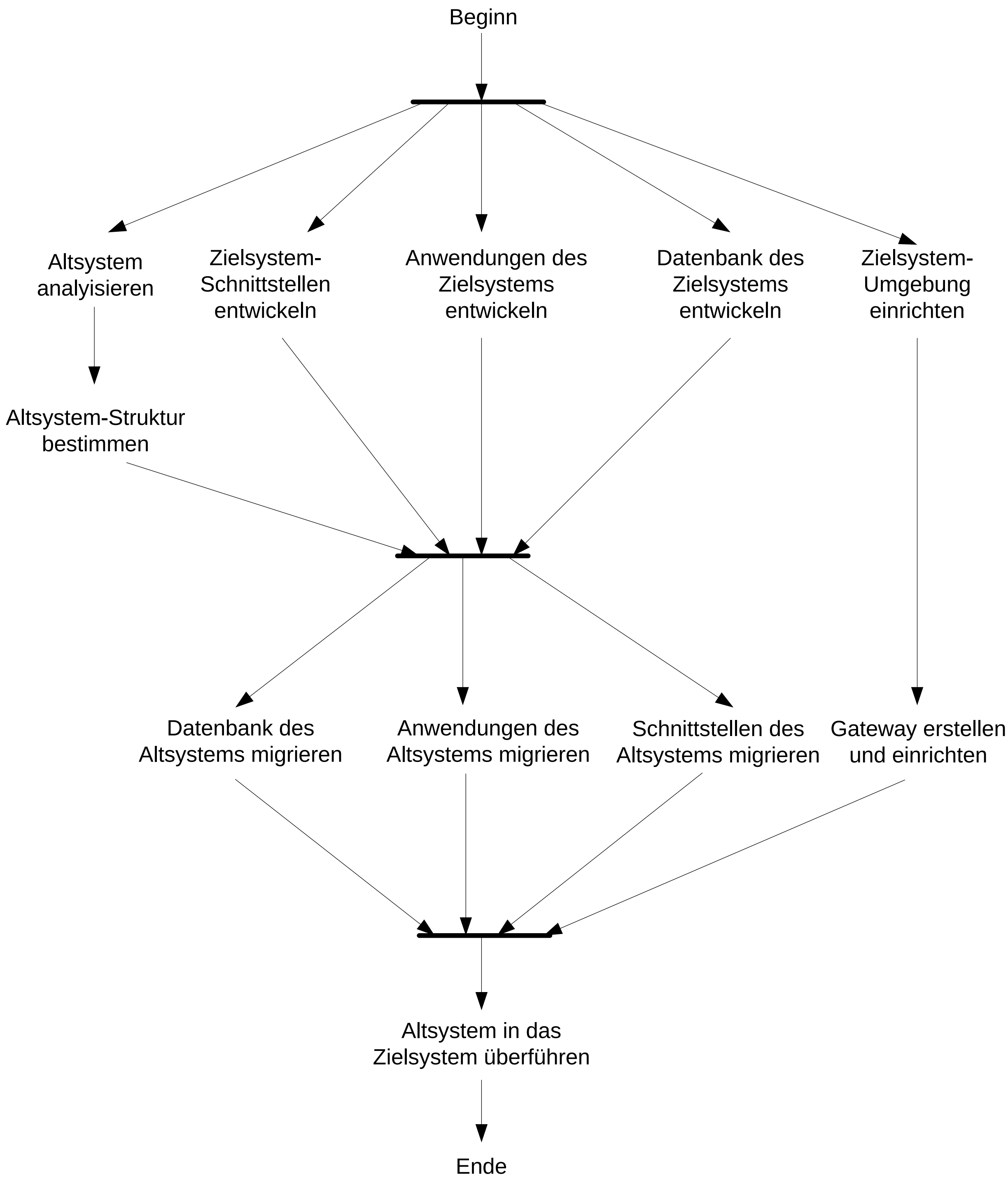


Abbildung 5 - Chicken Little-Iterationsphase sinngemäß nach (Brodie & Stonebraker 1995, S.59)

Auf die genannten Schritte wird an dieser Stelle im Detail eingegangen:

Altsystem analysieren: Im Zuge einer Migration wird das *Altsystem untersucht* und es werden dabei jene Komponenten identifiziert, die in das Zielsystem überführt werden müssen. Für diesen Zweck werden die im Altsystem vorhandenen Funktionen, Daten und Schnittstellen mit den Systemanforderungen des Zielsystems verglichen. Unnötige Bestandteile werden im weiteren

Migrationsprozess nicht weiter beachtet. Das Ergebnis der Analysephase hat daher großen Einfluss auf den Gesamtaufwand eines Migrationsprojekts.

Altsystem-Struktur bestimmen: In diesem Schritt wird die *Struktur des Altsystems* ermittelt. Dabei soll festgestellt werden, inwieweit sich das Altsystem in einzelne Bestandteile mit definierten Schnittstellen zerlegen lässt. Je klarer sich das System aufgliedern lässt, desto eher wird eine schrittweise Migration gelingen. Dabei muss jedoch beachtet werden, dass zwischen Systemteilen undokumentierte Abhängigkeiten bestehen können.

Zielsystem-Schnittstellen entwickeln: Sind die Schnittstellen des Altsystems bekannt, kann untersucht werden, welche Anknüpfungspunkte für umgestellte Komponenten zum Altsystem bestehen. Je nach Umstellungsart einer Komponente (siehe Abschnitt 3.3.1 - Funktionsmigration) resultieren unterschiedliche *Schnittstellen* zu anderen Systemen. Auf dieser Information des Altsystems aufbauend können Systemschnittstellen und Benutzeroberflächen *des Zielsystems* definiert werden.

Anwendungen des Zielsystems entwickeln: Für die zuvor definierten *Schnittstellen* werden nun die entsprechenden *Anwendungen* entwickelt (siehe Kapitel 3.3.2 - Funktionsmigration). Um das Risiko meist komplexer Migrationen nicht zu erhöhen, sollte auf neue Funktionalität verzichtet werden. Dies ist jedoch nach der Praxiserfahrung der Buchautoren Michael Brodie und Michael Stonebraker meist nicht möglich, da eine aufwendige und somit kostenintensive Systemumstellung vor dem Management häufig mit Zusatzfunktionalität gerechtfertigt werden muss (Brodie & Stonebraker 1995, S.9).

Datenbank des Zielsystems entwickeln: Bei der Entwicklung der *Datenbankstruktur des Zielsystems* wird das Datenbankschema des Altsystems in die Zieldatenbank übergeführt. Dabei sollten die Anforderungen nicht übererfüllt werden und bei der Schemadefinition nur benötigte Datenfelder berücksichtigt werden. Ziel dieser Maßnahme ist es, die Systemkomplexität möglichst gering zu halten.

Zielsystem-Umgebung einrichten: Anschließend wird für die entwickelten *Zielsystemkomponenten* eine *Testumgebung eingerichtet*. Diese stellt nach der Systemumstellung die Produktivumgebung dar.

Gateway erstellen und einrichten: Werden nur einzelne Komponenten des Altsystems in das Zielsystem umgestellt, find sowohl Zugriffe auf die Datenbank des Alt- als auch des Zielsystems statt. Ein *Gateway* ist eine zusätzliche Komponente, die dafür sorgt, dass Datenzugriffe entweder ausschließlich im Alt- oder im Zielsystem erfolgen. Dazu werden Zugriffe von speziellen Anwendungskomponenten, den sogenannten Gateways, abgefangen und umgeleitet. Diese werden in Kapitel 3.3.1 - Datenmigration näher erläutert.

Datenbank des Altsystems migrieren: Bevor *Daten* des Altsystems *migriert* werden können, müssen sie in das Format der Zieldatenbank umgewandelt werden; ungültige Daten werden dabei berichtigt. Zuvor sollte allerdings geprüft werden, ob der gesamte Datenbestand übernommen werden muss oder Teile davon nach einer etwaigen Archivierung verworfen werden können.

Anwendungen des Altsystems migrieren: Die entwickelten *Anwendungen* werden nun auf die Zielsystemumgebung *migriert*. Der Gateway ermöglicht es, dass neue und im Altsystem belassene Anwendungen auf dieselbe Datenbank zugreifen können (siehe Kapitel 3.3.1 - Datenmigration).

Schnittstellen des Altsystems migrieren: *Benutzeroberflächen und Systemschnittstellen*, deren zugehörige Anwendungen bereits auf das Zielsystem umgestellt wurden, werden in diesem Schritt ebenfalls *migriert*.

Altsystem in das Zielsystem überführen: Den *Abschluss einer Migrationsphase* bildet das Umschalten des installierten Testsystems auf ein Produktivsystem. Dabei werden zunächst Funktionalität und Daten identifiziert, die von den Anforderungen nicht erfasst sind und somit verworfen werden können. Daraufhin werden die geplante Struktur und die Umgebung des Zielsystems bestimmt, die zur Ausarbeitung eines Migrationsplans benötigt wird.

In Abbildung 5 werden die Abhängigkeiten der Vorgehensschritte untereinander veranschaulicht. Obwohl manche Schritte als parallelisierbar dargestellt sind, können diese nicht zu 100 Prozent abgegrenzt werden. Sie können daher nicht vollständig unabhängig voneinander ausgeführt werden. So basiert die Entwicklung der *Anwendungen des Zielsystems* teilweise auf dem Ergebnis des Vorgangs *Analyse des Altsystems*, diese sind jedoch als gleichzeitig durchführbar dargestellt.

3.2.3 Butterfly-Methode

Die Entwickler der Butterfly-Methode sehen die Datenmigration als Schlüssel zum Migrationserfolg (Bisbal u. a. 1999). Die Entwicklung der Zielsystemfunktionalität ist bei dieser Vorgehensweise vom restlichen Migrationsprozess abgekoppelt und kann unabhängig von der Datenmigration erfolgen. Mehrere Softwarekomponenten ermöglichen bei der Butterfly-Methode, den gesamten Datenbestand im laufenden Betrieb in das Zielsystem zu übertragen. Die Vorgehensweise ist in Abbildung 6 skizziert, wobei die einzelnen Schritte in Kapitel 3.3.1 - Datenmigration näher beschrieben werden. Sobald sich der Großteil der Daten in der Zieldatenbank befindet, werden nach Abschalten des Altsystems die verbleibenden Daten umkopiert. Nach Abschluss des Kopiervorgangs wird das parallel zur restlichen Migration entwickelte Zielsystem aktiv geschaltet.

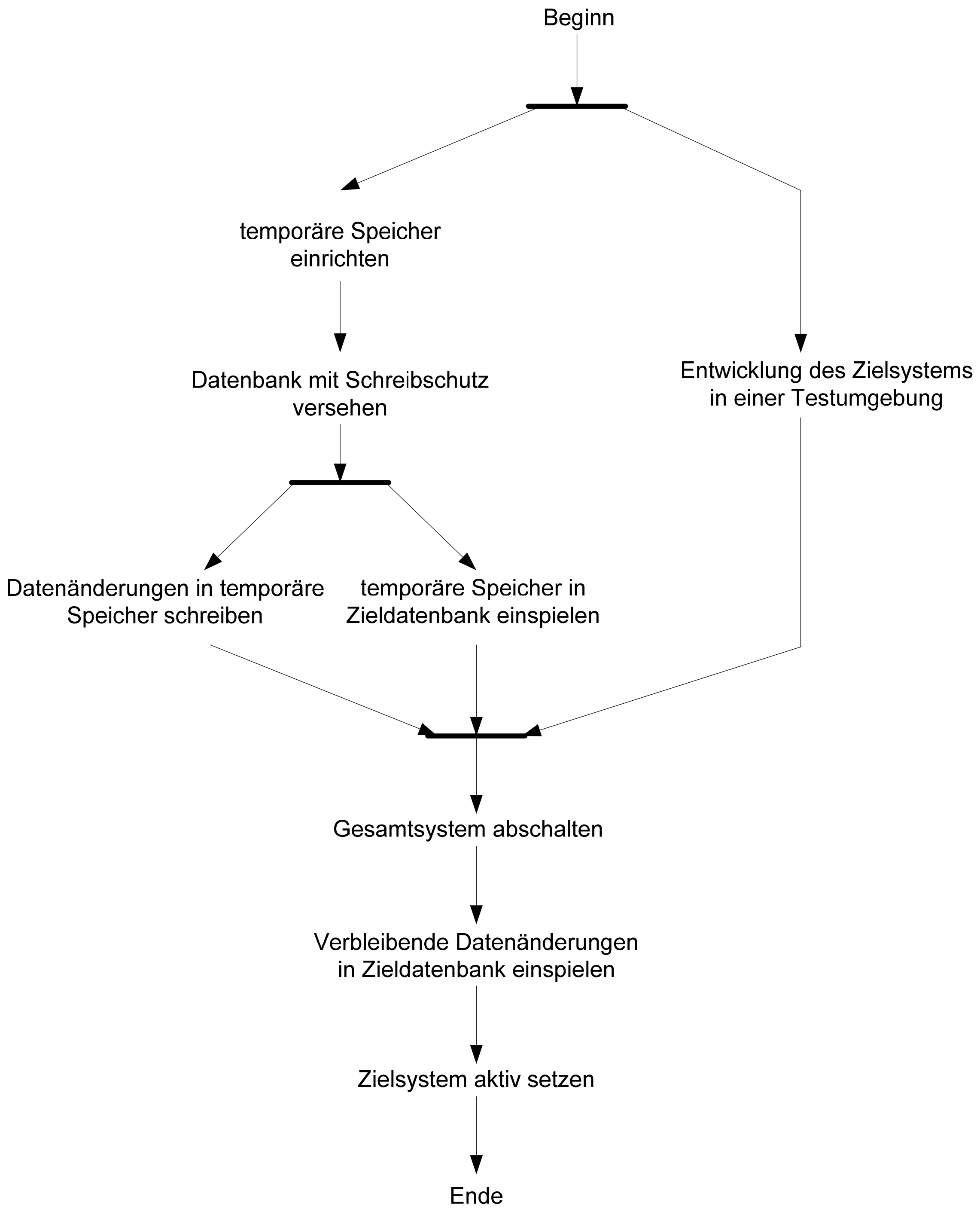


Abbildung 6 - Ablauf der Butterfly-Migration

3.2.4 Geschäftsprozessbasierte Migrationsmethode

Als Basis einer Migration können auch Geschäftsprozesse dienen, auf deren Basis Anwendungen entwickelt werden. Da das Zielsystem die Struktur eines verteilten Systems besitzt, können Komponenten lose miteinander gekoppelt werden (Bing u. a. 1997).

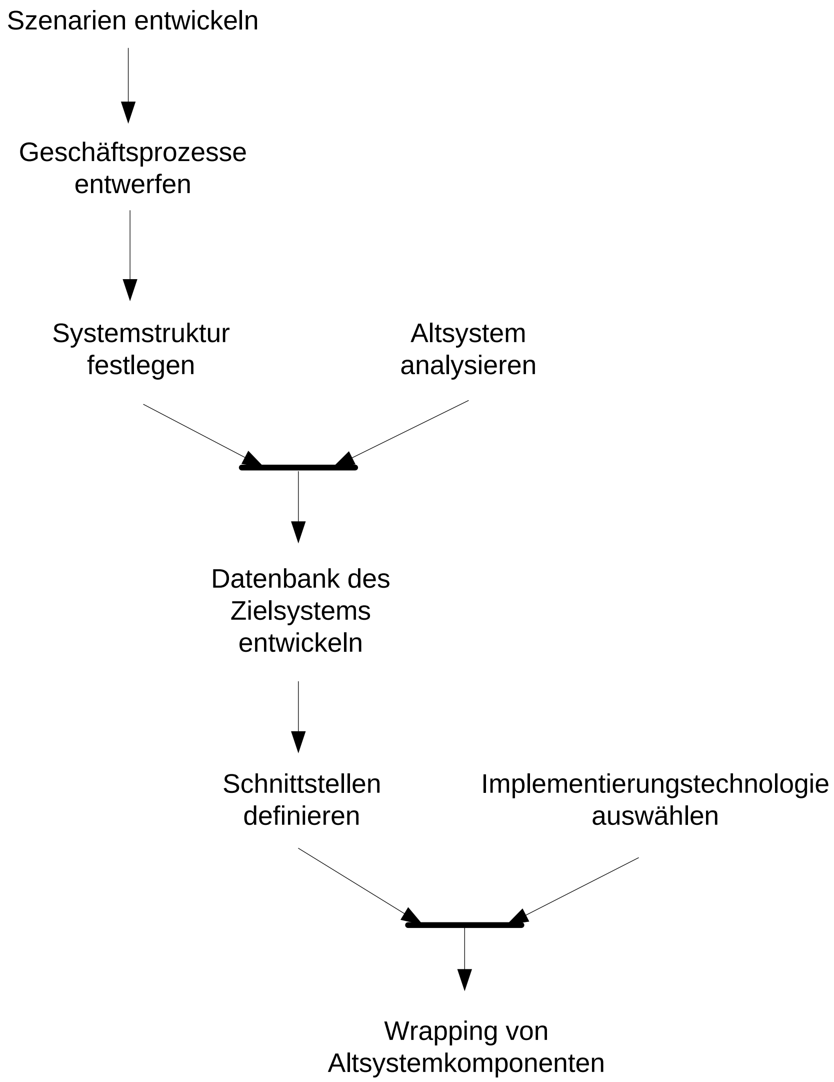


Abbildung 7 - Ablauf geschäftsprozessbasierte Migration in Anlehnung an (Juric u. a. 2000, S.35)

Basierend auf den definierten Abläufen wird die Struktur des Zielsystems festgelegt. Dabei wird versucht, bestehende Komponenten einzubinden. Deshalb werden jene Funktionen und Daten aus dem Altsystem identifiziert, die sich für eine Zielsystem-Integration eignen. Um Altsystemkomponenten in das Zielsystem einbinden zu können, müssen diese meist um Systemschnittstellen ergänzt werden. Derartige Erweiterungen werden als Wrapping bezeichnet und in Kapitel 3.3.2 - Funktionsmigration näher beschrieben.

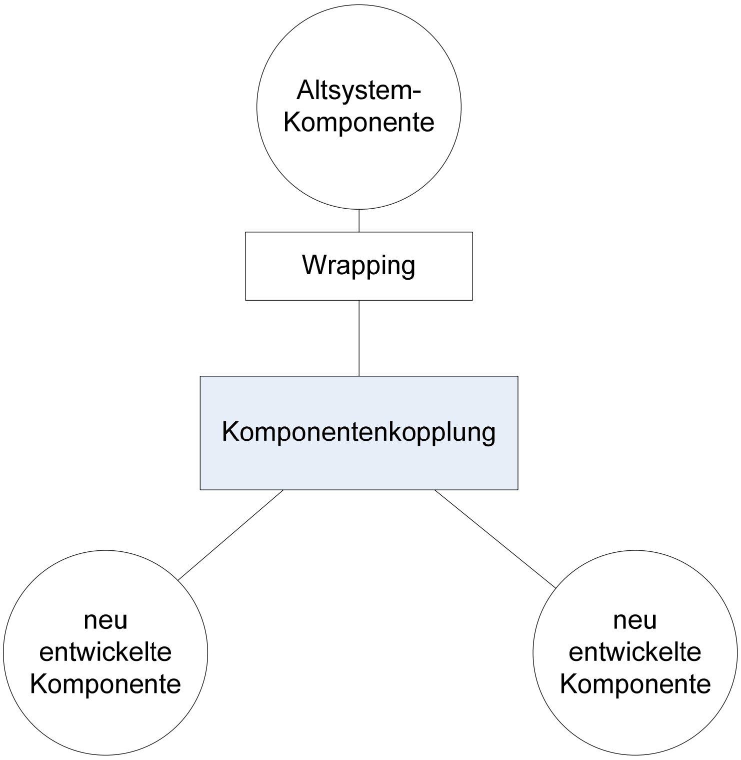


Abbildung 8 - Aufbau eines verteilten Systems

Zur Kommunikation zwischen den Komponenten (siehe Abbildung 8) sollte eine standardisierte Vermittlungsschnittstelle wie die Object Request Broker (ORB)-Architektur genutzt werden. So bietet die Common Object Request Broker Architecture (CORBA) eine plattformübergreifende ORB-Implementierung. Diese ist für eine Vielzahl von Programmiersprachen und Betriebssystemen verfügbar und erleichtert somit die Integration in heterogenen Systemumgebungen (Juric u. a. 2000, S.36). CORBA hat sich jedoch nicht durchgesetzt und wird heute kaum noch verwendet. Sie wurde von der Webservice-Technologie abgelöst. Webservices bieten zwar im Gegensatz zum ORB keine klassischen Middleware-Funktionen wie verteilte Transaktionen, sondern sie setzen auf eine reine Punkt-zu-Punkt-Kommunikation. Aufgrund der einfachen Handhabung und einer umfangreichen Tool-Unterstützung werden Webservices jedoch wesentlich breiter zur losen Kopplung von Komponenten eingesetzt. Einige der genannten Aspekte kommen auch in der Definition von Papazoglou und Georgakopoulos zum Ausdruck.

"Services are self-describing, open components that support rapid, low-cost composition of distributed applications" (Papazoglou & Georgakopoulos 2003, S.26)

Webservices stellen daher besonders für moderne heterogene Systemumgebungen das Mittel der Wahl dar.

3.2.5 Individueller Migrationsprozess

Manche Fachautoren vertreten die Ansicht, dass aufgrund der Vielzahl unterschiedlicher Migrationsprojekte der Einsatz standardisierter Vorgehensweisen unzulässig ist (Stevens & Pooley

1998). So sind einige zu migrierende Excel-Makros nicht mit einer umfangreichen und jahrzehntealten Cobol-Anwendung vergleichbar. Außerdem sind Vorgehensmodelle schwierig zu evaluieren, da der durchschnittliche Softwareentwickler nur über wenig Migrationserfahrung verfügt. Firmen veröffentlichen Fallstudien über Systemumstellungen äußerst selten und meist nur dann, wenn sie erfolgreich waren.

Um die genannten Probleme zu vermeiden wird der Einsatz sogenannter Systems Re-Engineering Patterns vorgeschlagen, die mit Entwurfsmustern für Softwarearchitekturen vergleichbar sind. Dabei handelt es sich um bewährte Expertenlösungen, um eine genau definierte Problemstellung zu lösen. Ein Systems Re-Engineering Pattern besteht somit aus einer Problembeschreibung, dem Lösungsvorschlag und einer Auflistung der Vor- und Nachteile dieses Vorgehens. Die Problembeschreibung enthält neben den formalen Anforderungen weitere Vorgaben wie Budgetbeschränkungen, Wünsche der künftigen Systemanwender und den geforderten Fertigstellungstermin.

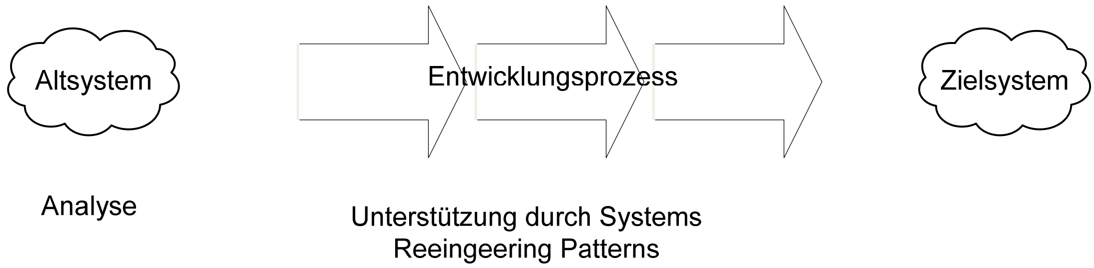


Abbildung 9 - Systems Re-Engineering Patterns in Anlehnung an (Stevens & Pooley 1998, S.20)

Im Zuge eines Migrationsprojekts können mithilfe der Analyse des Altsystems mehrere Teilaufgabenstellungen identifiziert werden. Für diese werden im nächsten Schritt geeignete Systems Re-Engineering Patterns gesucht, deren Empfehlungen gefolgt werden kann. Da diese im Gegensatz zu vollständigen Vorgehensmodellen klein und somit überschaubar sind, kann die Eignung für ein konkretes Projekt auch von Entwicklern mit fehlender Migrationserfahrung ermittelt werden.

3.2.6 Gegenüberstellung der Vorgehensweisen

Wie in Abbildung 10 ersichtlich ist, nehmen bei den Umstellungsprozessen Cold Turkey und Chicken Little sowohl Daten als auch Funktionen eine wichtige Rolle ein. Im Gegensatz dazu stellt die Butterfly-Methode das Überspielen der Daten vom Alt- in das Zielsystem in den Vordergrund und entkoppelt die Funktionsmigration vollständig vom Rest des Umstellungsprozesses. Wird eine geschäftsprozessbasierte Migration durchgeführt, liegt der Fokus auf den Funktionen eines Altsystems, die als isolierte Komponenten in das Zielsystem integriert werden sollen (Bing u. a. 1997).

Migrationsprozesse können nach dem Risiko eines Systemausfalls klassifiziert werden. So kommt bei der Cold Turkey-Systemumstellung (Big Bang-Migration) ein zuvor nicht in der Praxis erprobtes Zielsystem zum Einsatz. Eine schrittweise Migration hat demgegenüber den Vorteil, dass durch

vereinzelte Schwierigkeiten nicht das gesamte Zielsystem unbrauchbar wird, sondern schlimmstenfalls die Änderungen einer Migrationsphase fehlschlagen. Beim Chicken Litte-Ansatz sind demgegenüber alle Migrationsschritte reversibel, weshalb das Risiko eines längeren Systemausfalls minimal ist. Das gleiche gilt für die Butterfly-Methode, da die Datenmigrationsphase jederzeit gestoppt beziehungsweise die folgende Funktionsmigration zurückgenommen werden kann.

Bei einem verteilten Zielsystem, das aus einer geschäftsprozessbasierten Migration hervorgeht, können Fehler in voneinander gekapselten Komponenten einem bestimmten Teil des Systems zugeordnet und somit einfacher als bei einer Cold Turkey-Umstellung korrigiert werden. Im beschriebenen Ansatz (Bing u. a. 1997) sind jedoch keine Maßnahmen zur Risikominimierung der Systemumstellung vorgesehen, weshalb diese Methode in der Übersichtstabelle als risikoreich eingestuft wird. Kommt ein individueller Umstellungsprozess zum Einsatz, ist es empfehlenswert, zu Beginn die Höhe des Migrationsrisikos zu ermitteln und mit den Migrationsanforderungen abzustimmen.

Migrationsart	**sowohl Daten als auch Funktionen wesentlich**	**Fokus auf Datensicht**	**Fokus auf Funktionssicht**	**geringes Migrationsrisiko**
Cold Turkey (Big Bang)	✓			
Chicken Little (schrittweise Migration)	✓			✓
Butterfly-Methode		✓		✓
Geschäftsprozess -basierte Migration			✓	

Abbildung 10 - Gegenüberstellung der Migrationsarten

3.3 Migrationsschritte

Der Migrationsablauf kann in die Datenmigration und die Funktionsmigration unterteilt werden.

3.3.1 Datenmigration

Sind Altsysteme durchgehend in Betrieb und müssen sie auch während einer Systemumstellung unterbrechungsfrei funktional bleiben, stellt die Migration der Daten eine große Herausforderung dar. Während der Chicken Litte-Ansatz den Datenbestand dadurch konsistent hält, dass sich alle Daten entweder im Alt- oder im Zielsystem befinden, setzt die Butterfly-Migration Abstraktionskomponenten ein, um Ortstransparenz bei einem verteilten Datenbestand zu ermöglichen.

Zunächst wird die Verwendung von Gateways bei der **Chicken Little-Migration** beschrieben. Das Aufbrechen des Migrationsprozesses in eine Reihe überschaubarer Schritte ist verbunden mit einer

aufwendigen Datenverwaltung. Werden Funktionen in Etappen migriert, müssen entweder neue Funktionen auf die alte Datenbasis oder Funktionen des Altsystems auf die neue Datenbasis zugreifen.

Wenn bei einer iterativen Migration zuerst die Datenbank auf das Zielsystem umgestellt wird, müssen sowohl Altsystem- als auch Zielsystemfunktionen eine gemeinsame Zielsystemdatenbasis verwenden. Der in Abbildung 11 dargestellte und für diese Aufgabe zuständige *Forward Gateway* leitet dazu Datenzugriffe der noch nicht migrierten Komponenten des Altsystems auf die Datenbank des Zielsystems um.

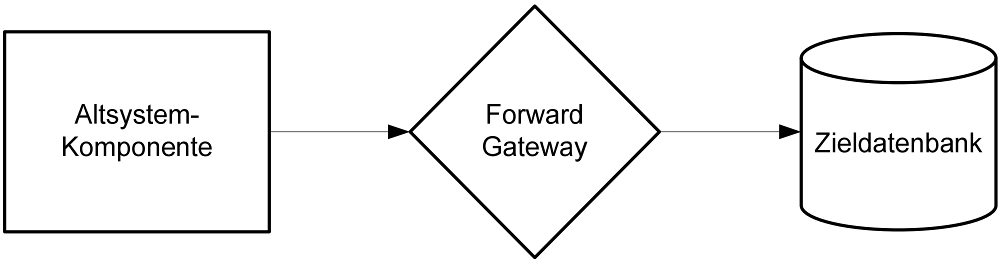


Abbildung 11 - Forward Gateway zur Aufrufumleitung auf Zielsystemdatenbank sinngemäß nach (Brodie & Stonebraker 1995, S.24)

Werden hingegen zunächst einzelne Funktionen in das Zielsystem überführt und die Datenbank im Altsystem belassen, kommt ein *Reverse Gateway* (siehe Abbildung 12) zum Einsatz. Dieser ermöglicht Datenzugriffe von Funktionen des Zielsystems auf die Datenbasis des Altsystems.

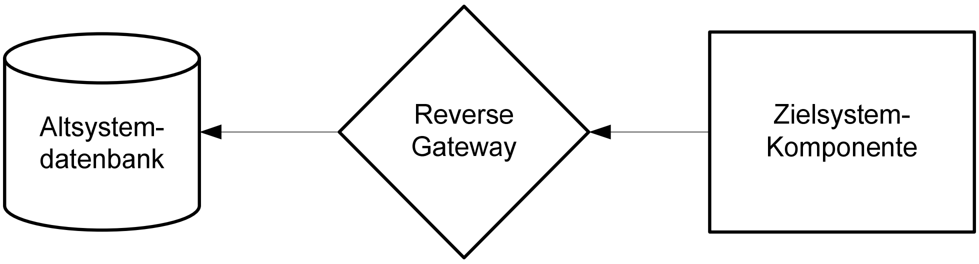


Abbildung 12 - Reverse Gateway zur Aufrufumleitung auf Altsystemdatenbank sinngemäß nach (Brodie & Stonebraker 1995, S.24)

Gateways sind vor allem in Hinblick auf Transaktionen stark eingeschränkt. So ist es bei abgebrochenen Transaktionen schwierig, die Datenkonsistenz zwischen Alt- und Zielsystem sicherzustellen. Weichen die Schemata der migrierten Systeme stark voneinander ab, so steigt der Aufwand zur Koordination des Datenzugriffs rapide an. Gateways sind daher komplizierte Komponenten, deren Aufbau und Betrieb schwierig ist.

Die **Butterfly-Migration** (Abbildung 13) setzt mehrere Komponenten ein, um den Datenbestand im laufenden Betrieb in die Zieldatenbank überspielen zu können. Aufgabe des *data access allocator* ist es, bei Abfragen die Daten vom richtigen Speicherort zu lesen. Außerdem fängt er während der Migration schreibende Zugriffe ab und hält die vorgenommenen Änderungen in

Behelfsdatenstrukturen, sogenannten *tempstores*, fest. Gleichzeitig wird der vorhandene Datenbestand überspielt. Der Chrysaliser verpackt dazu während des Betriebs im Altsystem befindliche Daten in tempstores. Diese werden anschließend nach dem first in first out (FIFO)-Warteschlangenprinzip in die Zieldatenbank eingespielt. Gemäß dem FIFO-Prinzip werden, vergleichbar mit einer Supermarktkasse, die ältesten tempstores zuerst abgearbeitet.

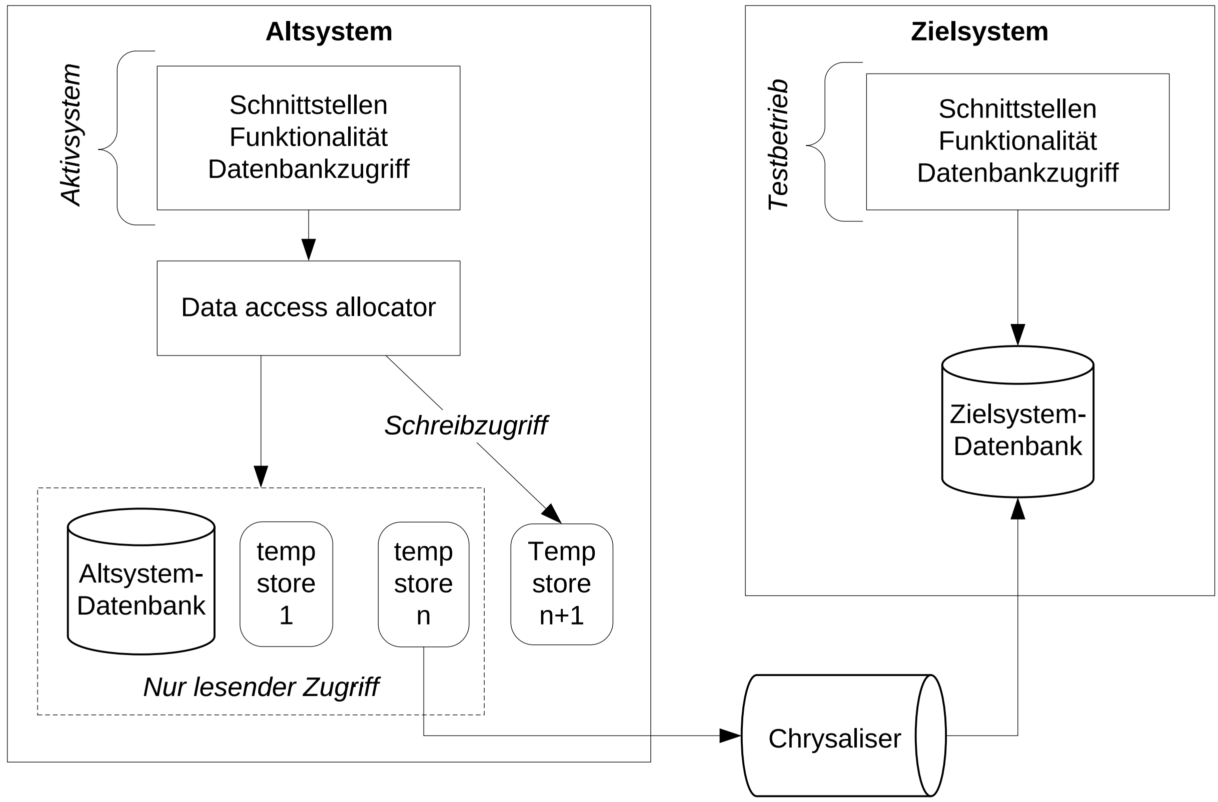


Abbildung 13 - Datenmigration bei der Butterfly-Methode in Anlehnung an (Bing u. a. 1997, S.317)

Sobald sich keine Daten mehr im Altsystem befinden und die Warteschlange der zu persistierenden tempstores eine festgelegte Anzahl unterschritten hat, wird das Altsystem abgeschaltet. Daraufhin hat der Chrysaliser Gelegenheit, alle verbleibenden tempstores in die Zieldatenbank einzuspielen und somit die Zieldatenbank zu vervollständigen. Während des gesamten Prozesses befinden sich daher zu keinem Zeitpunkt dieselben Daten sowohl im Alt- als auch im Zielsystem. Das System ist wieder einsatzbereit, sobald das unabhängig von der Datenmigration entwickelte Zielsystem hochgefahren wurde.

3.3.2 Funktionsmigration

Um Funktionen des Altsystems an aktuelle Anforderungen anzupassen und in ein Zielsystem überzuführen, existieren unterschiedliche Strategien. Diese können, abhängig vom Änderungsaufwand am Altsystem, klassifiziert werden, wobei bei einer System-Evolution ein System weitgehend beibehalten wird und eine System-Revolution mit einer Vielzahl an Änderungen einhergeht. In

Abbildung 14 sind die drei Strategien Wrapping, Migration und Neuentwicklung zur Anpassung von Altsystemen mit ihrem jeweiligen Systemeinfluss in Beziehung gesetzt.

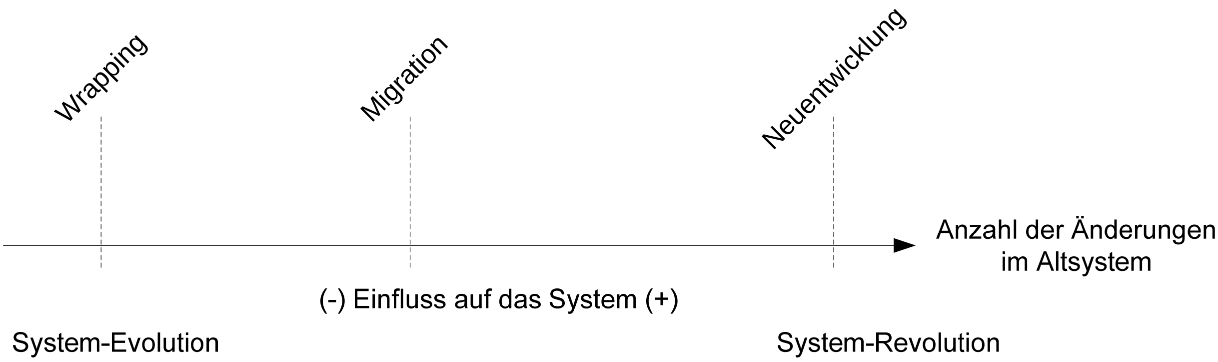


Abbildung 14 - Umstellungsstrategien von Altsystemen adaptiert von (Bisbal u. a. 1999, S.3)

Wrapping bezeichnet das Verpacken einer Komponente in eine neue Komponente, die mehr Schnittstellen zur Außenwelt zur Verfügung stellt. Vorhandene Funktionen und Daten der Altkomponente werden beibehalten und um Zugriffsmöglichkeiten ergänzt, wodurch sich nur wenige Änderungen am Altsystem ergeben. Die erweiterte Komponente kann als Server betrachtet werden, dessen Funktionalität für die von ihm abhängigen Systemteile, den Clients, über standardisierte Systemschnittstellen, wie beispielsweise Webservices, abrufbar ist.

Bei einer *Migration* wird vorhandene Funktionalität in eine neue Systemumgebung übertragen. Voraussetzung dafür ist, den Systemaufbau zu kennen, da meist der Quelltext des Altsystems angepasst werden muss, um in der Zielumgebung lauffähig zu sein. Ein typisches Migrationsbeispiel ist die Portierung einer Anwendung auf ein anderes Betriebssystem.

Eine *Neuentwicklung* setzt vorhandene Funktionalität unter Zuhilfenahme aktueller Technologie von Grund auf neu um. Das bedeutet, dass bestehende Algorithmen adaptiert und in eine neue Systemumgebung übernommen werden. Diese Methode ermöglicht es zwar, das System den Anforderungen optimal anzupassen, sie versursacht jedoch den meisten Aufwand.

Die Entscheidung für eine Umstellungsstrategie muss nicht auf Systemebene getroffen werden. Stattdessen wird typischerweise für alle Systemkomponenten getrennt festgelegt, ob Wrapping, Migration oder Neuentwicklung zum Einsatz kommen soll (Bisbal u. a. 1999, S.3).

3.4 Migrationszeitpunkt

Bei einer Umstellung nach dem *cut-and-run*–Prinzip wird zunächst das Altsystem deaktiviert und daraufhin das neu entwickelte Zielsystem aktiv gesetzt. Bei einer *phasenweisen Umstellung* geschieht dieses Umschalten in kleinen Schritten. Dabei werden inkrementell einzelne Komponenten des Altsystems in das Zielsystem übergeführt. Fällt die Wahl auf einen *parallelen Betrieb*, so sind

während der Umstellung alle Funktionen des Altsystems und des Zielsystems aktiv. Das Zielsystem wird so lange im Praxiseinsatz getestet, bis es als uneingeschränkt tauglich befunden wird. Erst wenn das System einwandfrei läuft, wird das Altsystem deaktiviert (Bisbal u. a. 1999, S.6)

4 Integration des eTutors in Moodle

Im Zuge der Diplomarbeit soll das intelligente tutorielle System *eTutor* in das Learning Management System *Moodle* integriert werden (vgl. Kapitel 1.2 - Zielsetzung). Die Erstellung des Zielsystems soll durch eine geeignete Migrationsstrategie unterstützt und das Ergebnis hinsichtlich Zielerreichungsgrad und Erweiterungspotential beurteilt werden.

4.1 Migrationskonzept

Die Integration des eTutor in Moodle setzt eine geeignete Vorgehensweise zur Umstellung voraus. Dafür werden unterschiedliche Aspekte der in Kapitel 3 beschriebenen Migrationskonzepte zu einem individuellen Migrationsprozess kombiniert. Da die Umstellung auf die neue Lernumgebung während der vorlesungsfreien Zeit erfolgen kann, muss das System nicht durchgehend verfügbar sein. Ein vorübergehender Systemausfall stellt kein Problem dar. Aus diesem Grund ist eine Systemumstellung auf einen Schlag, wie sie die Cold Turkey-Migration beschreibt, praktikabel. Aufwändige Maßnahmen zur Sicherung eines laufenden Systembetriebs, wie sie bei der Butterfly-Migration vorgesehen sind, werden nicht benötigt.

Eine Neuentwicklung des Zielsystems von Grund auf wird nicht angestrebt. Vielmehr soll ein möglichst großer Teil der existierenden Expertenmodulfunktionalität in das Zielsystem übernommen werden. Für diesen Zweck eignet sich der im Chicken Litte-Migrationsprozess beschriebene Ablauf zur schrittweisen Migration von Systemteilen. Dieser allgemein beschriebene Migrationsprozess wurde auf die speziellen Erfordernisse der eTutor-Migration angepasst, woraus folgendes Vorgehen resultiert:

1. Systemanalyse
2. Anwendungen des Zielsystems entwickeln
3. Datenbank des Zielsystems entwickeln
4. Datenbank des Altsystems migrieren

Die Chicken Little-Migration sieht vor, diese Prozessschritte wiederholt zu durchlaufen, um mit jeder Migrationsphase weitere Komponenten zu migrieren. Da bei der eTutor-Migration funktionierende Expertenmodule zum Einsatz kommen und es möglich ist, das migrierte System in der vorlesungsfreien Zeit intensiv zu testen, kann die Umstellung des eTutor-Systems auf einmal erfolgen. Somit wird der oben beschriebene Ablauf nur einmal ausgeführt. Dennoch bleibt ein Restrisiko, da das Zielsystem während der vorlesungsfreien Zeit nicht im Echtbetrieb getestet werden kann.

4.2 Migrationsdurchführung

Die Systemimplementierung erfolgt gemäß den in Abschnitt 4.1 genannten Migrationsschritten.

4.2.1 Systemanalyse

Ziel der Systemanalyse ist es, das Altsystem zu analysieren und benötigte Funktionalität und Daten zu identifizieren. Dazu werden in einem ersten Schritt die Komponenten des Altsystems, also die Lernumgebungen eTutor und Moodle, hinsichtlich Aufbau und Funktionsumfang betrachtet. Als Informationsquellen konnten die textuelle Dokumentation und die Lernsystem-Entwickler dienen. Da der eTutor am Institut für Wirtschaftsinformatik – Data and Knowledge Engineering entwickelt wurde, war es mir möglich, Fragen direkt an die Programmierer zu richten. Die Moodle-Entwickler sind in Internet-Diskussionsforen aktiv und geben dort Hilfestellungen. Als Student und Tutor am DKE-Institut konnte ich aus erster Hand erfahren, wie *eTutor*-Anwender Übungsbeispiele bearbeiten. Der mitbetreuende Assistent meiner Diplomarbeit, Christian Eichinger, verwendet zudem eTutor in der Rolle eines Lehrveranstaltungsleiters und war in der Lage, mir zielführende Hinweise hinsichtlich dieses Anwendungsbereichs zu geben. Zur Analyse des exakten Aufbaus der Moodle-Administrationskomponente diente eine lokale Testinstallation.

4.2.1.1 Funktionalität

Im Folgenden wird das Altsystem aus Anwendersicht beschrieben. Mehrere Personengruppen verwenden *Funktionalität* des Zielsystems: Studenten benutzen die Lernumgebung zum Wissenserwerb, während sich Kursleiter und Tutoren um die Bereitstellung der Lerninhalte, so wie die Korrektur von Übungsbeispielen, kümmern. Administrationsdialoge ermöglichen dabei das Anlegen, Ändern und Löschen der Beispiele, wobei der Kursleiter für die Bereitstellung von Tests zuständig ist, die ein oder mehrere Übungsbeispiele umfassen. Zudem werden Funktionen zur Sicherung und Wiederherstellung der Beispieldatenbank benötigt.

Übungsbeispiele können in automatisch- oder von Hand-korrigierte Beispiele kategorisiert werden. Für automatisch korrigierte Beispiele kann das Lernsystem eine Rückmeldung erstellen, während manuell bewertete Übungsbeispiele von Tutoren beurteilt werden. Jede der Übungsabgaben wird einem Tutor zugeteilt, der sich die Studentenabgabe zur Beurteilung auf seinen Computer herunterladen kann. Am DKE-Institut wird der Großteil der Übungsausarbeitungen im PDF-Format abgegeben, die mit Softwareprodukten wie *Adobe Acrobat* mit Kommentaren versehen werden können. Die erbrachte Leistung wird mittels einer Punktzahl bewertet und dem Student wird seine kommentierte Abgabe bereitgestellt. Der Ablauf einer manuellen Übungskorrektur ist in Abbildung 15 dargestellt.

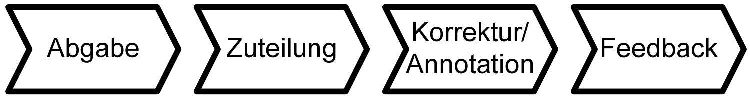


Abbildung 15 - Ablauf Tutorenkorrektur

Im Folgenden werden benötigte Anwendungsfälle des Altsystems aufgezeigt, die im Zielsystem ebenfalls verfügbar sein müssen. Diese sind in einem Use Case-Diagramm in Abbildung 16 dargestellt.

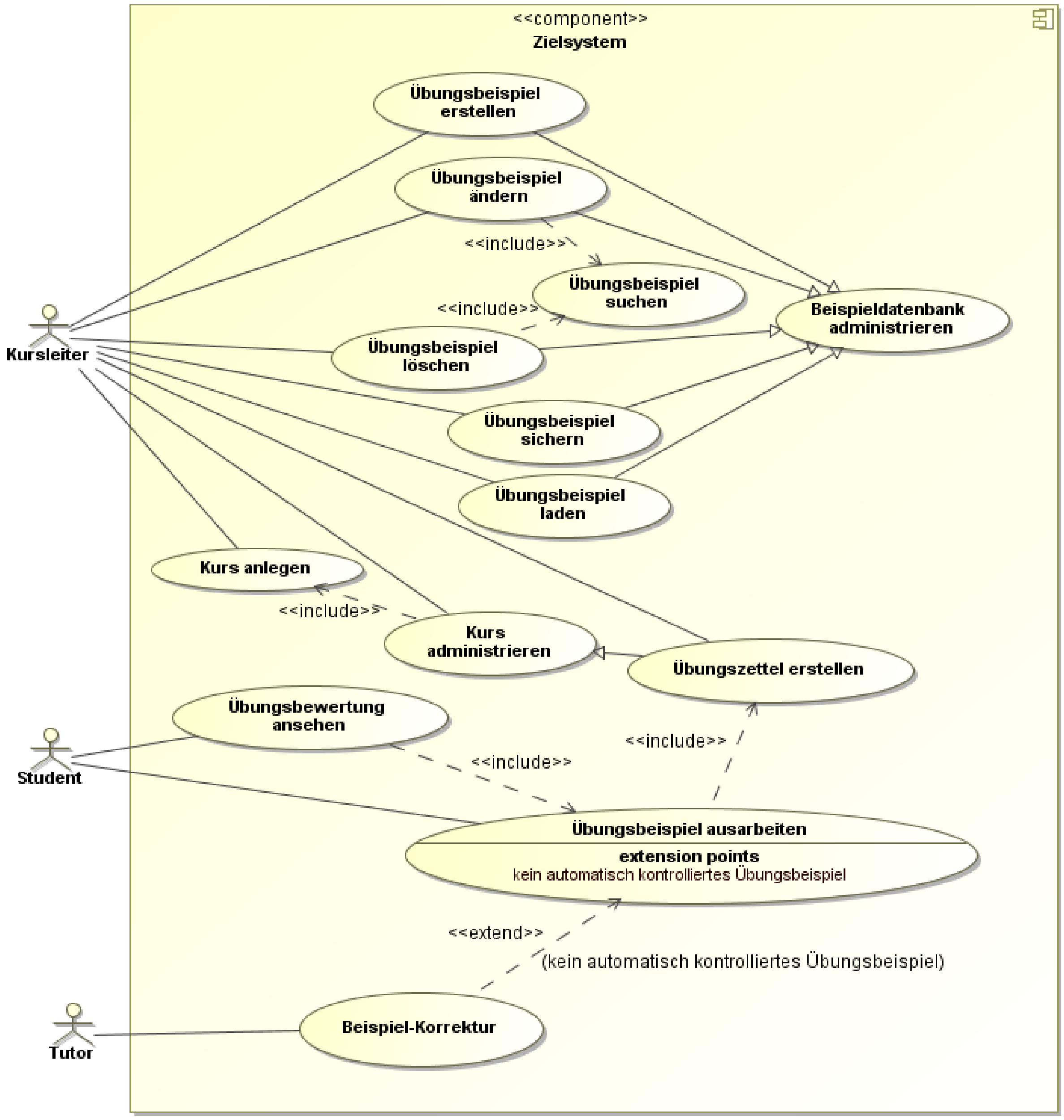


Abbildung 16 - Anwendungsfälle des Zielsystems

Das Use Case-Diagramm folgt der UML-Notation und ist wie folgt zu interpretieren: Die Anwendungsfälle *Übungsbeispiel erstellen*, *Übungsbeispiel ändern*, *Übungsbeispiel löschen*, *Übungsbeispiel sichern* und *Übungsbeispiel laden* sind vom Anwendungsfall *Beispieldatenbank administrieren* abgeleitet. Mit den Sicherungs- und Ladeoperationen kann die Beispieldatenbank in eine XML-Datei geschrieben und daraufhin in eine andere Moodle-Instanz eingespielt werden; zudem dienen sie zur Archivierung von Beispielangaben. Vor Änderungs- und Löschoperationen muss zunächst das entsprechende Übungsbeispiel gesucht werden, was durch die *include*-Beziehung in

Richtung Anwendungsfall *Übungsbeispiel suchen* zum Ausdruck gebracht wird. Damit ein Student seine *Übungsbewertungen ansehen* kann, muss er zunächst ein *Übungsbeispiel ausarbeiten*. Dies setzt wiederum voraus, dass vom Kursleiter ein *Übungszettel erstellt* wurde, in dem das jeweilige Beispiel eingestellt wurde.

Die funktionalen Anforderungen des Altsystems stimmen mit jenen des Zielsystems überein, weshalb versucht wird, bestehende Funktionalität unverändert in das Zielsystem zu übernehmen. Aus diesem Grund kann für eine detailliertere Erläuterung der Lernsystem-Funktionen auf die eTutor-Benutzeranleitung verwiesen werden[1]. Die Lernumgebung *Moodle* bietet Funktionen wie Multiple Choice-Übungsbeispiele und ein Diskussionsforum für Kursteilnehmer. Derartige Zusatzfunktionalität ist jedoch nicht Teil der Migration, weshalb in dieser Diplomarbeit nicht weiter darauf eingegangen wird.

Im Use Case-Diagramm werden unterschiedliche Arten von Beziehungen verwendet. Durchgehende Linien zeigen, welche Anwendungsfälle von einem bestimmten Akteur verwendet werden. Gestrichelte Linien werden über eine textuelle *extends*- oder *include*-Beschriftung in Spitzklammern näher beschrieben. Die *extends*-Anweisung erweitert im Unterschied zur *include*–Beziehung nur dann das Verhalten eines Anwendungsfalls, wenn die im *extension point* festgelegte Bedingung erfüllt ist. Im Zuge der Bearbeitung eines Übungsbeispiels nimmt der Tutor daher genau dann eine Beispiel-Korrektur vor, wenn es sich um ein manuelles Übungsbeispiel handelt.

4.2.1.2 eTutor

Vorrangiger Zweck des *eTutor*-Systems ist die automatische Korrektur von Übungsbeispielen, wofür sogenannte Expertenmodule zum Einsatz kommen. Ein Expertenmodul ist für einen abgegrenzten Themenbereich zuständig und beinhaltet Algorithmen zur Analyse und Bewertung von Übungsaufgaben; zudem können dynamisch generierte Rückmeldungen erstellt werden. Im Modulumfang sind außerdem Dialoge zur Bearbeitung der Musterlösung enthalten. Expertenmodule existieren beispielsweise für die Stoffgebiete SQL, JDBC und XQuery. Die automatische Prüfung wird jedoch fortlaufend erweitert. Zum Zeitpunkt der Diplomarbeits-Erstellung befand sich etwa im Rahmen einer anderen Diplomarbeit ein Bewertungsmodul für UML-Klassendiagramme in Entwicklung (Koenings 2010).

Die *eTutor-Core*-Komponente integriert Expertenmodule und implementiert Themengebiet-übergreifende Funktionalität. So besitzen Angaben aller Beispieltypen dasselbe Format und werden aus diesem Grund zentral in der eTutor-Core-Datenbank abgelegt. Dasselbe gilt für Kursteilnehmer, Zuteilungen und deren Bewertungen. Obwohl Übungsbearbeitungs-Dialoge zum Expertenmodul-Umfang gehören, stellt der eTutor-Core die Web-Benutzerschnittstelle zur Verfügung. Von Tutoren zu

[1] Diese befindet sich auf der beiliegenden CD im Verzeichnis /documentation/projektdokumentation/benutzerhandbuch_etutor.doc

kontrollierende Übungsaufgaben werden über ein Dateiupload-Formular hochgeladen. Tutoren werden bei Übungsbewertungen von einem speziellen Administrationsinterface unterstützt. Ein ideales ITS, das in Kapitel 2 beschrieben wird, besteht aus den Komponenten *Kommunikationsmodul*, *Studentenmodul*, *Unterrichtsmodul* und *Expertenmodul*. Im eTutor wird der aktuelle Lernfortschritt des Studenten nicht dafür genutzt, um speziell angepasste Skripten und Übungsbeispiele vorzuschlagen. Stattdessen kommuniziert das *Unterrichtsmodul* direkt mit dem *Kommunikationsmodul*, dem eTutor-Core.

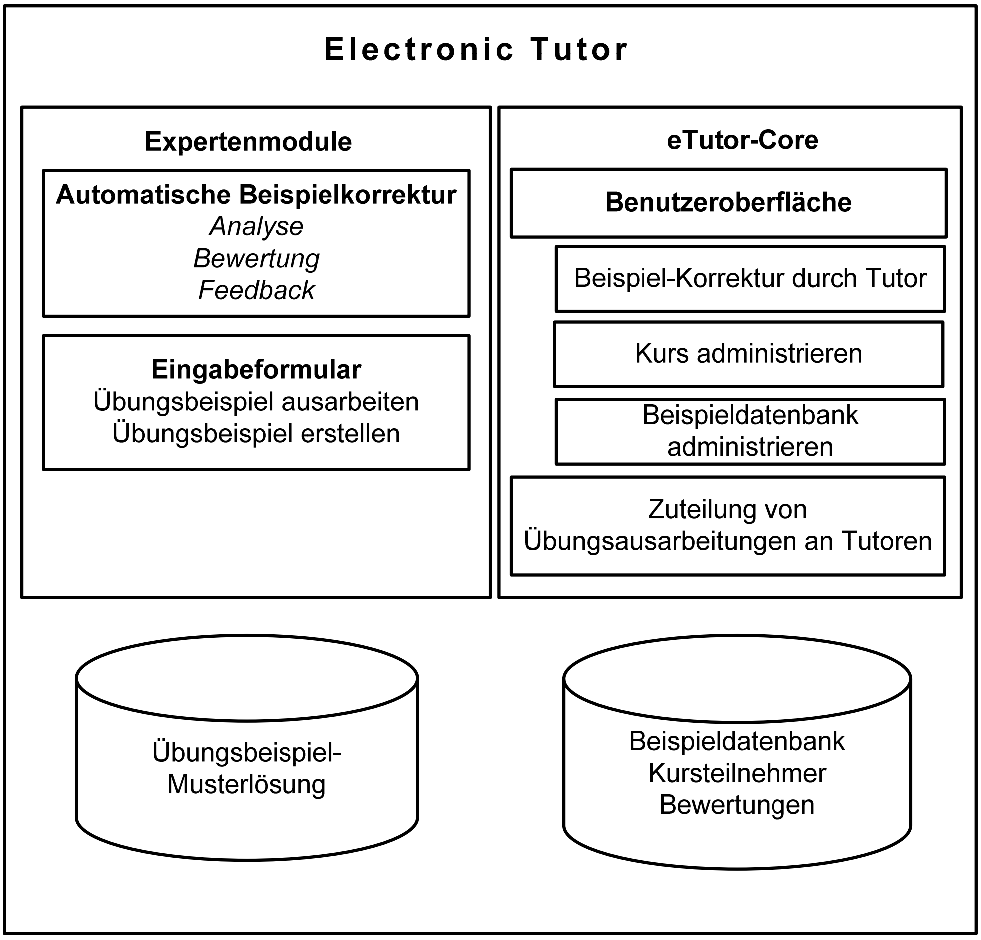


Abbildung 17 - Analyse des Altsystems: eTutor

Das eTutor-System bietet daher folgende Funktionalität

Funktion	Anmerkung
Beispieldatenbank administrieren	Beispielübergreifende Administrationsfunktionalität liegt im eTutor-Core, während beispielbezogene Übungsdialoge Teil der Expertenmodule sind.
Kurs anlegen	Das Anlegen eines Kurses stellt eine Beispiel-übergreifende Funktionalität dar und gehört daher zum eTutor-Core.
Übungszettel erstellen	Übungszettel können im eTutor-Core erstellt werden.

Übungsbeispiel ausarbeiten	Der Eingabedialog zum Bearbeiten von Übungsaufgaben wird je nach Beispieltyp unterschiedlich gestaltet. Die Funktionalität zur Ausarbeitung von Übungsbeispielen ist daher in den Expertenmodulen implementiert.
Übungsbewertung ansehen	Alle Übungsabgaben und deren Bewertungen sind im eTutor-Core abgelegt. Soll eine Übungsabgabe angezeigt werden, wird diese von einem Dialog innerhalb des Expertenmoduls dargestellt.
Beispiel-Korrektur durch Tutor	Die Ausarbeitung und Bewertung manuell korrigierter Übungsaufgaben findet ausschließlich im eTutor-Core statt.
automatische Beispielkorrektur	Die Korrekturfunktionalität ist in den Expertenmodulen abgelegt.

Abbildung 18 - eTutor-Funktionalität des Altsystems

4.2.1.3 Moodle

Das Learning Management System *Moodle* verwendet sogenannte Kurse als eine Art Container für Lerninhalte. Kurse können neben (multimedial unterstützten) Skripten auch simple Übungsaufgaben enthalten. Die Interaktion zwischen Studenten wird durch Diskussionsforen ermöglicht. Den Zugriff von Kursteilnehmern, Kursleiter und Tutoren reglementiert Moodle über ein umfangreiches Rollenmodell, das es ermöglicht, Zugriffsrechte feingranular zu vergeben. Derzeit sind knapp 50 000 Moodle-Installationen weltweit namentlich registriert (Moodle Statistics 2010). Aufgrund unterschiedlichster Benutzeranforderungen sind die Moodle-Entwickler um hohe Interoperabilität mit anderen Systemen und um gute Erweiterbarkeit bemüht. Demgemäß kann der Moodle-Login beispielsweise auf *Active Directory*-Benutzerkonten zugreifen, um existierende Logindaten nutzen zu können. Die Erweiterbarkeit von Moodle wird mittels Plugin-Schnittstelle sichergestellt.

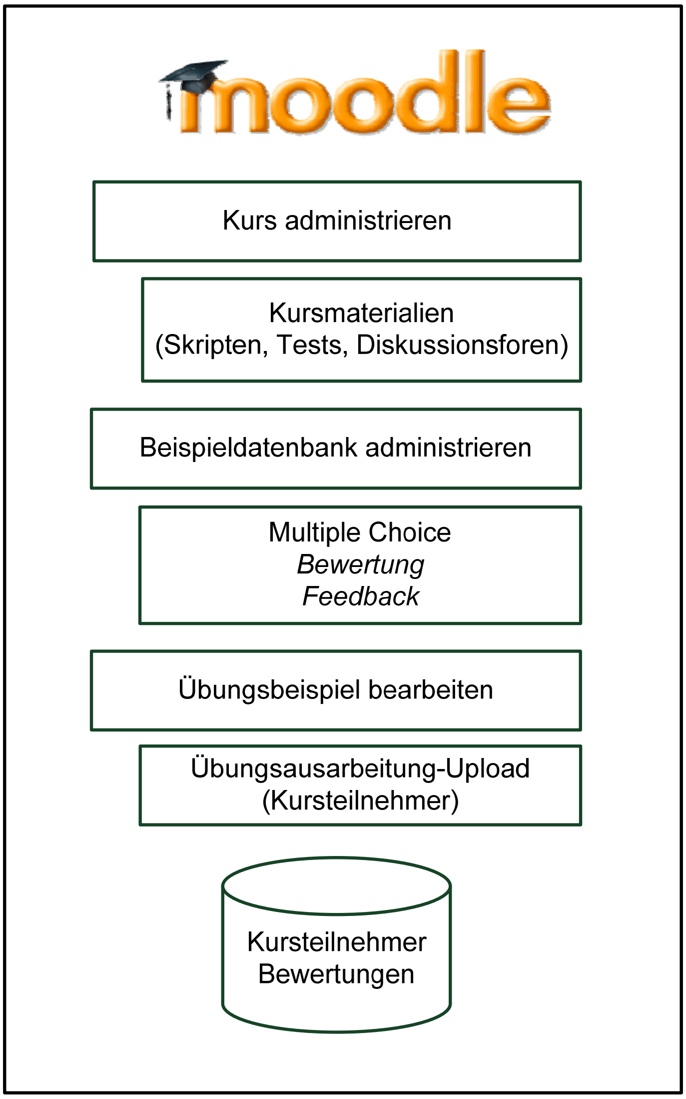


Abbildung 19 - Analyse des Altsystems: Moodle

Die Datenbank von Moodle in Version 1.95 besteht aus 214 Tabellen, deren Name aus einem frei wählbaren Präfix und einer Bezeichnung des Inhalts zusammengesetzt ist. In den folgenden Beschreibungen wird zur Steigerung der Übersichtlichkeit auf die Nennung des Präfixes verzichtet.

Innerhalb des Datenbankschemas, die Moodle-Installation der Diplomarbeit verwendet die Datenbank *MySQL*, bestehen Verknüpfungen zwischen Tabellen nur implizit. Das bedeutet, dass in der Datenbank keine Foreign Keys angelegt werden, sondern diese ausschließlich in der XML-Beschreibung der Tabellenstruktur definiert werden. Diese Beschreibung ist unabhängig von der konkreten Datenbankimplementierung, wodurch die Austauschbarkeit des DBMS sichergestellt werden soll. Um die Zugriffe einzuschränken, ist in Moodle ein umfangreiches Rollenkonzept implementiert.

In Abbildung 20 sind jene Moodle-Funktionen angeführt, die für den Aspekt der eTutor-Integration relevant sind.

Funktion	**Beschreibung**
Beispieldatenbank administrieren (nur einfache Beispieltypen)	Moodle unterstützt das Anlegen, Ändern und Löschen einfacher Übungsbeispiele wie Multiple-Choice-Aufgaben.
Kurs anlegen	Für jede Lehrveranstaltung ist ein Kurs notwendig, um Studenten Skripten und Übungszettel zur Verfügung stellen zu können. Die Bewertungen der Kursteilnehmer sind ebenfalls Teil des jeweiligen Kurses.
Übungszettel erstellen	Ein Übungszettel umfasst die von Kursteilnehmern zu bearbeitenden Übungen.
Übungsbeispiel ausarbeiten (nur einfache Beispieltypen)	Moodle stellt Korrekturfunktionalität für einfache Übungsbeispiele bereit.
Übungsbewertung ansehen	Die bei der Übungsausarbeitung erreichten Punktestände können tabellarisch angezeigt werden.
automatische Beispielkorrektur (nur einfache Beispieltypen)	Einfache Übungsbeispiele, für die nur ein Vergleich der Übungsabgabe des Studenten mit der Musterlösung notwendig ist, kann Moodle automatisch korrigieren. Weicht die Übungsabgabe von der voreingestellten Lösung ab, kann Moodle keine Hinweise zur Verbesserung geben.

Abbildung 20 - Moodle-Funktionalität des Altsystems

4.2.1.4 Zielsystem-Funktionalität

Die Schwächen des eTutor-Altsystems sind nicht in seiner Beispielkorrektur-Funktionalität sondern im Bereich der Verwaltung von Kurs- und Teilnehmerdaten zu finden. So müssen sich Studenten zur Nutzung des eTutors selbständig beim System registrieren und zum entsprechenden Kurs einschreiben. Da keine Funktion zur Entfernung und zur Archivierung alter Kursdaten aus der Datenbank implementiert ist, wächst der Datenbestand fortlaufend an. Die mit hohem Aufwand entwickelten eTutor-Expertenmodule entsprechen demgegenüber voll den Anforderungen an eine automatisierte Korrektur von Übungen. Aus diesem Grund sollen diese mit möglichst wenigen Anpassungen in das Zielsystem übernommen werden können.

Einen weiteren Kritikpunkt am bisherigen eTutor-System stellt neben der fehlenden Möglichkeit, Übungsangaben und deren Bewertungen mehrsprachig anzubieten, die unzeitgemäße Benutzerschnittstelle dar. Speziell für Austauschstudenten wäre es wünschenswert, Übungsaufgaben nicht nur in deutscher sondern auch in englischer Sprache bearbeiten zu können. Außerdem entspricht die Weboberfläche des eTutor-Systems nicht mehr dem aktuellen Stand der Technik. Ein Frame-basiertes Design gilt als überholt und die eTutor-Oberfläche ist mit aktuellen Versionen des Webbrowsers Internet Explorer inkompatibel.

Das Learning Management System *Moodle* wird zentral für die Johannes Kepler Universität vom E-Learning- und Evaluierungsservice administriert. Der Betrieb liegt somit außerhalb des Einflussbereichs des DKE-Instituts. Das Moodle-System wird dabei in unregelmäßigen Abständen

aktualisiert, wobei diese Möglichkeit auch nach der Migration bestehen bleiben soll. Im Zuge der Integration dürfen keine Änderungen am Moodle-Kernsystem vorgenommen werden, da diese auf jede einzuspielende Moodle-Version portiert werden müssten. Als Alternative bietet sich die Verwendung der von Moodle bereitgestellten Plugin-Schnittstelle an. Damit kann das Moodle-System an vorbereiteten Anknüpfungspunkten um eigene Funktionalität ergänzt werden.

In folgender Übersicht werden die Komponenten des Zielsystems beschrieben:

Funktion	Umstellungs-strategie	Anmerkung
Beispieldatenbank administrieren	Migration	Die Aufgabe der Verwaltung von Übungsbeispielen soll vom eTutor-Core in das Moodle-System übertragen werden. Die Moodle-Funktionalität zum Hinzufügen, Editieren und Löschen einfacher Übungsbeispiele sollte um eine Unterstützung für eTutor-Beispiele erweitert werden. Diese wird mittels Plugin realisiert.
Kurs anlegen	-	Kurse werden im Zielsystem von Moodle angelegt. Die in Moodle eingebaute Funktion zum Erstellen von Kursen entspricht jener des eTutor-Core. Diese eTutor-Funktionalität kann daher ohne Änderungen durch jene von Moodle ersetzt werden.
Übungszettel erstellen	Migration	Übungsbeispiele, die per Zufallsgenerator aus einem Beispielpool ausgewählt werden, weist Moodle bei jedem Abgabeversuch neu zu. Damit ein Student seine einmal erhaltenen Übungsbeispiele behält, muss beim Erstellen eines Übungszettels darauf geachtet werden, die Einstellung *Anzahl Abgabeversuche* auf den Wert *Eins* zu setzen. Das eTutor-System lässt trotz dieser Einstellung eine beliebige Anzahl an Übungsbewertungen zu.
Übungsbeispiel ausarbeiten	Migration	Moodle stellt die zur Übungsausarbeitung benötigte Benutzeroberfläche bereit. eTutor übernimmt die automatische Korrektur von Übungsabgaben.
Übungsbewertung ansehen	Migration	Im Zielsystem sind Übungsbewertungen im Moodle-System gespeichert. Da die automatische Beispielprüfung durch den eTutor erfolgt, muss die vom System vergebene Punktzahl zu Moodle übertragen werden.
Beispiel-Korrektur durch Tutor	Neuentwicklung	Diese Funktionalität soll mittels Moodle-Erweiterung vollständig neu umgesetzt werden. Im Altsystem wurde pro Übungszettel jedem Kursteilnehmer ein Tutor zugewiesen. Da nicht von allen Studenten Übungen ausgearbeitet wurden, konnte es zu einer unfairen Verteilung des Korrekturaufwands für die Tutoren kommen. Aus diesem Grund wird im Zielsystem die Tutoren-Zuteilung erst nach Abgabeschluss vorgenommen, wodurch Übungsabgaben auf alle Tutoren gleichmäßig verteilt werden können.
automatische Beispielkorrektur	Wrapping	Im Zielsystem stellt Moodle die Benutzerschnittstelle zur Verfügung. Um die Korrekturfunktionalität in den eTutor-Expertenmodulen beibehalten zu können, müssen die dafür

	benötigten Benutzerdialoge zum Moodle-System übertragen werden.

Abbildung 21 - Zielsystem-Funktionalität

4.2.2 Anwendungen des Zielsystem entwickeln

Die Schnittstellen des Zielsystems sind in Abbildung 22 ersichtlich: Studenten und Kursleiter greifen mittels Webbrowser auf die Moodle-Webseite zu. Sobald ein eTutor-Übungsbeispiel aufgerufen wird, sendet Moodle eine Webservice-Anfrage an das eTutor-System, wo die Anfrage vom dem Beispieltyp entsprechenden Expertenmodul verarbeitet wird.

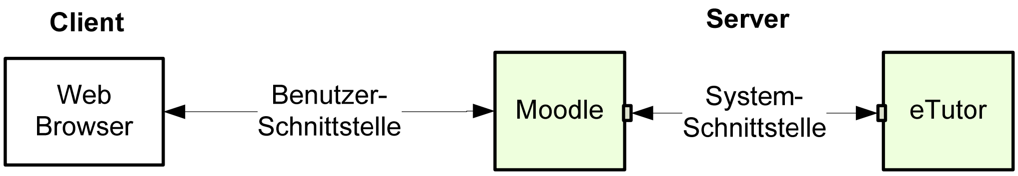


Abbildung 22 - Zielsystem-Schnittstellen

Im Altsystem sind Benutzerdialoge zum Erstellen und Bearbeiten von Übungsbeispielen in den jeweiligen eTutor-Expertenmodulen abgelegt. Da Lernende im Zielsystem nicht mehr direkt auf den eTutor sondern auf das zwischengeschaltete Moodle-System zugreifen (siehe Abbildung 22), müssen die Administratordialoge und Eingabemasken aus den Expertenmodulen im Moodle-System verfügbar gemacht werden. Für diesen Zweck werden bei jedem Beispielzugriff Übungsbeispiel-bezogene Benutzerdialoge von eTutor zum Moodle-System übertragen. Es wäre auch möglich gewesen, diese Benutzerdialoge fest im Moodle-System abzulegen, was jedoch einen größeren Änderungsaufwand verursacht hätte. Außerdem verbleibt durch die Dialog-Übertragung die Erstellung und Ausarbeitung der Übungsbeispiele im Expertenmodul und somit innerhalb der Kontrolle des DKE-Instituts. Die Kapselung von Funktionen und Dialogen im Expertenmodul wurde daher einer festen Dialogeinbindung in Moodle vorgezogen.

Um die im eTutor-System liegenden Beispieldialoge in Moodle verfügbar zu machen, können diese entweder mittels Frame in die Moodle-Oberfläche eingebunden oder per Webservice übertragen werden. Für die Frame-Variante spricht der geringe Umstellungsaufwand, da die Benutzeroberfläche des Altsystems bereits Frame-basiert ist. Demgegenüber ist jedoch die Datenübertragung zwischen Frames des eTutor- und des Moodle-Systems problematisch; der Zugriff auf eine gemeinsame Session wäre nur über Umwege möglich. Zudem ist die Verwendung von Frames nicht mehr zeitgemäß, weshalb diese Integrationsart als ungeeignet einzustufen ist (Rabin & McCathieNevile 2008). Mittels Webservices können in verschiedenen Technologien (beispielsweise PHP und Java) implementierte Komponenten miteinander kommunizieren und somit, gemäß dem Prinzip der geschäftsprozessorientierten Migration (siehe Kapitel 3.2.4), in einen Gesamtablauf integriert werden. Für die lose Kopplung von Lernmodulen unterschiedlicher Lernumgebungen werden Webservices als

besonders geeignet erachtet (Brooks u. a. 2006), weshalb sie auch zur Integration des eTutor in Moodle eingesetzt wurden. Die Webservice-Technologie wird in Kapitel 4.3.3.1 - Webservice-Technologie näher beschrieben.

4.2.2.1 Beispielerstellung

Der Erstellungs- und Bearbeitungsprozess von Übungsbeispielen ist in Abbildung 23 in Form eines Sequenzdiagramms dargestellt. Im ersten Bearbeitungsschritt (1) werden vom Benutzer Beispieltyp-übergreifende Einstellungen vorgenommen. Eine Datenübertragung findet dabei ausschließlich zwischen Webbrowser und Moodle (2) statt, da diese Einstellungen im Moodle-System verwaltet werden. Im Gegensatz dazu sind die Beispieltyp-bezogene Musterlösung samt deren Benutzerdialogen Teil des eTutor-Expertenmoduls. Sobald der Benutzer die Bearbeitung fortsetzt (3) wird eine Anfrage an den eTutor geschickt (4). Da Beispieldialoge im eTutor-System verwaltet werden, müssen diese im Zuge der Lösungsbearbeitung mittels Webservice zur Moodle-Webschnittstelle übertragen werden (5). Im Dialog eingebundene Dateien (beispielsweise Bilder), werden bei der ersten Dialogverwendung im Moodle-System zwischengespeichert, damit diese nicht bei jedem Dialog-Aufruf erneut mittels Webservice zum Moodle-System übertragen werden müssen.

eTutor-seitig wird zu Beginn jedes Bearbeitungsprozesses eine Benutzersession angelegt (6). Abhängig vom Beispieltyp wird daraufhin in einem oder mehreren Benutzerdialogen die Musterlösung des neuen Übungsbeispiels erfasst. Der dafür notwendige Kommunikationsablauf (7-10) wird daher so lange wiederholt, bis alle Beispieldaten erfasst wurden (im Diagramm durch das Schlüsselwort *loop* gekennzeichnet). Nachdem das Übungsbeispiel erfolgreich in der Expertenmodul-Datenbank abgelegt wurde, werden nach einer Bestätigung des Benutzers (11) die neu erstellte Übung in der Moodle-Beispieldatenbank gespeichert und die eTutor-Benutzersession geschlossen (12).

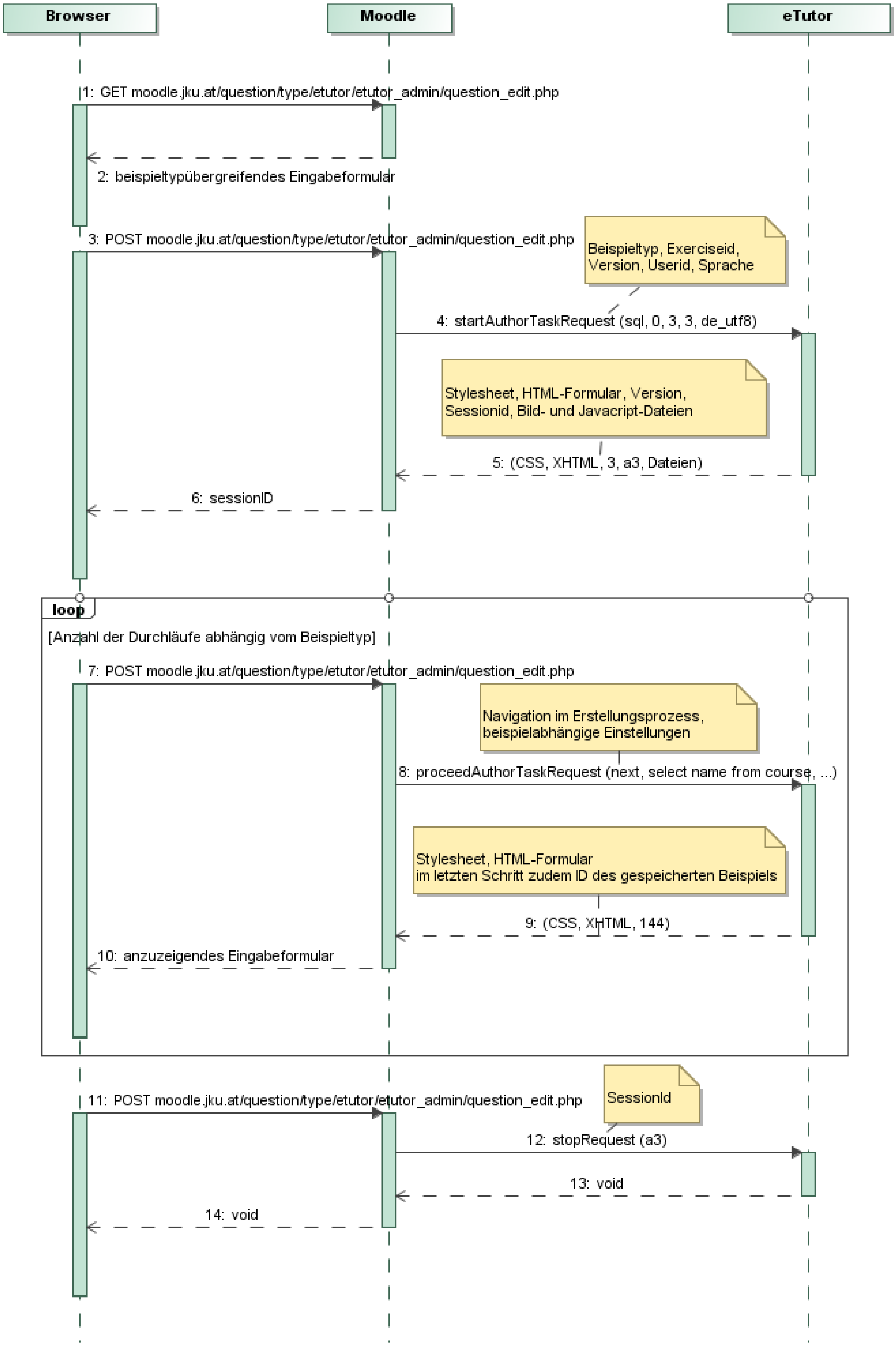


Abbildung 23 - Kommunikationsablauf bei der Beispielerstellung

4.2.2.2 Beispielbearbeitung

Die Bearbeitung eines Übungsbeispiels durch Kursteilnehmer ist dem Beispielerstellungsprozess (siehe Kapitel 4.2.2.1 - Beispielerstellung) sehr ähnlich. Die Dialoge zur Übungsausarbeitung sind ebenfalls im eTutor-Expertenmodul abgelegt, wobei diese je nach Beispieltyp unterschiedlich gestaltet sind. So enthalten SQL-Übungsbeispiele ein Textfeld zum Absetzen einer Datenbankabfrage, während bei einer relationalen Algebra Operatoren (beispielsweise für Projektion) mithilfe eines Auswahldialogs eingefügt werden können.

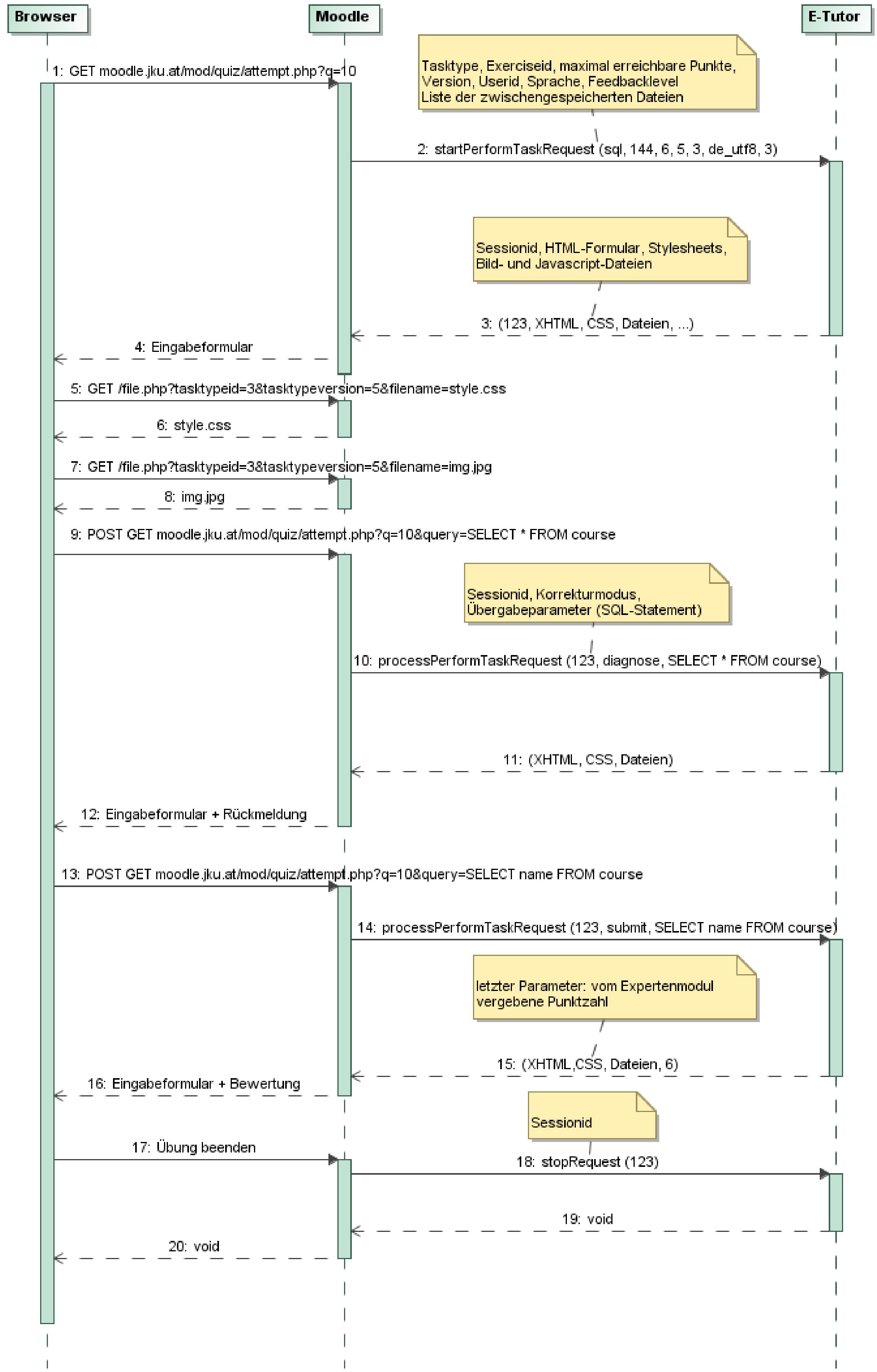


Abbildung 24 - Kommunikationsablauf bei der Beispielbearbeitung

Sobald die Schnittstellen von Moodle und eTutor feststehen, können die dafür benötigten Systemkomponenten entwickelt werden. Abbildung 25 stellt eine Übersicht aller beteiligten Komponenten dar, die auf Basis der festgelegten Schnittstellen in Kapitel 4.3 - Implementierung näher beschrieben werden.

Benutzer greifen mittels http-Schnittstelle auf das Learning Management System *Moodle* zu, welches sich wiederum aus dem Core-System und mehreren Erweiterungen (Plugins) zusammensetzt. Das Plugin *Tutoren-Bewertung* unterstützt die manuelle Korrektur von Übungsabgaben durch Tutoren. Mithilfe der Moodle-Erweiterungen *eTutor-Aufgabentyp* und *Beispiel-Administration* wird das Quiz-Modul dahingehend erweitert, dass Übungsbeispiele aus dem intelligenten tutoriellen System *eTutor* mithilfe des Webservice-Protokolls *SOAP* in Moodle eingebunden werden können. eTutor-seitig stellt *eTutor-Dispatcher* einen Webservice-Zugriff zur Verfügung und ermöglicht dadurch die Nutzung von Expertenmodul-Funktionalität. Expertenmodule können dazu entweder direkt oder, wenn sich diese auf einem anderen Computer befinden, mittels RMI-Schnittstelle in den eTutor-Dispatcher eingebunden werden.

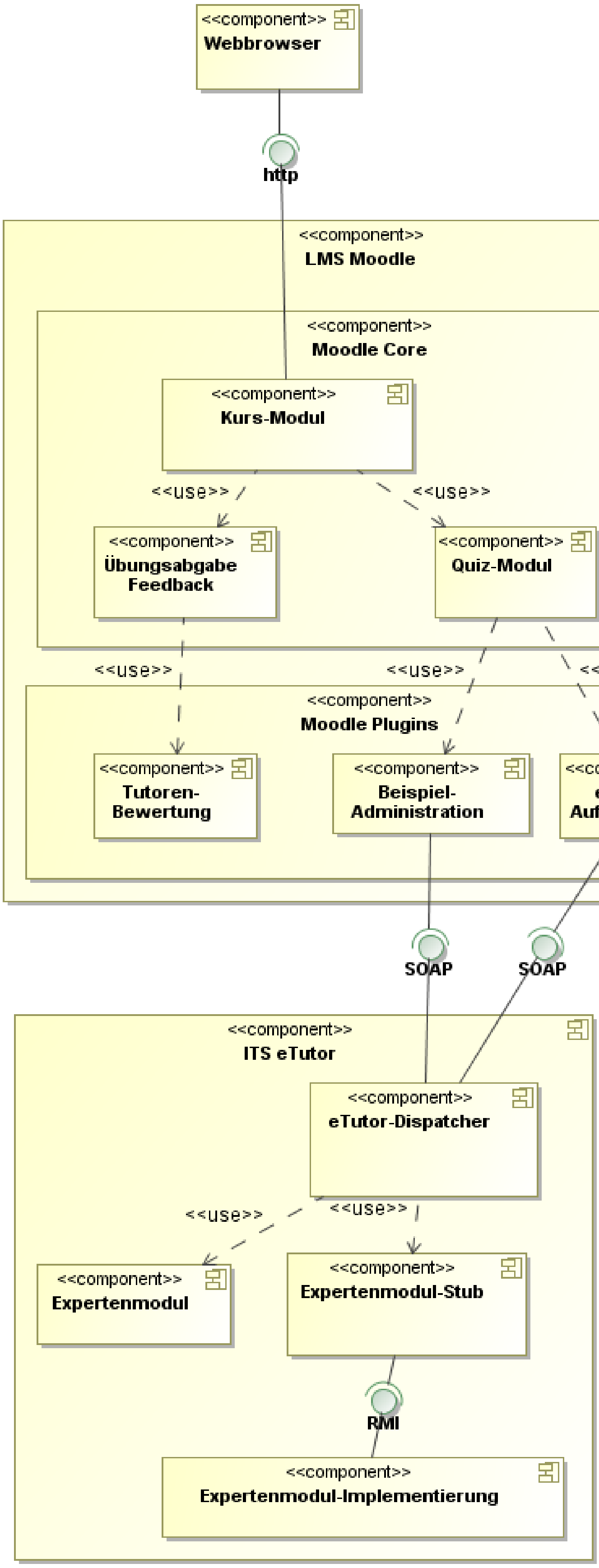


Abbildung 25 - Zielsystemarchitektur

4.2.3 Datenbank des Zielsystems entwickeln

In Abbildung 26 ist die Tabellenstruktur des Moodle-Plugins *eTutor-Aufgabentyp* zur Integration des eTutors dargestellt. Die Verbindungsdaten zum eTutor-Dispatcher werden in der Tabelle *question_etutor_server* abgelegt. Übungsbeispiele sind im Kernsystem gespeichert (*question*), diese werden jedoch über das *ID*-Fragenattribut um eTutor-spezifische Informationen erweitert (*question_etutor*). Angabetexte zu Übungsbeispielen (*question_etutor* und *question*) können in verschiedenen Sprachen (*question_etutor_internation* und *question_etutor_language*) bereitgestellt werden. Die Tabelle *question_etutor_tasktype* dient zur Kategorisierung von Übungsbeispielen nach Themengebieten. *Question_etutor_exerc_group* verweist pro Übungsbeispiel auf das entsprechende Expertenmodul im eTutor-System.

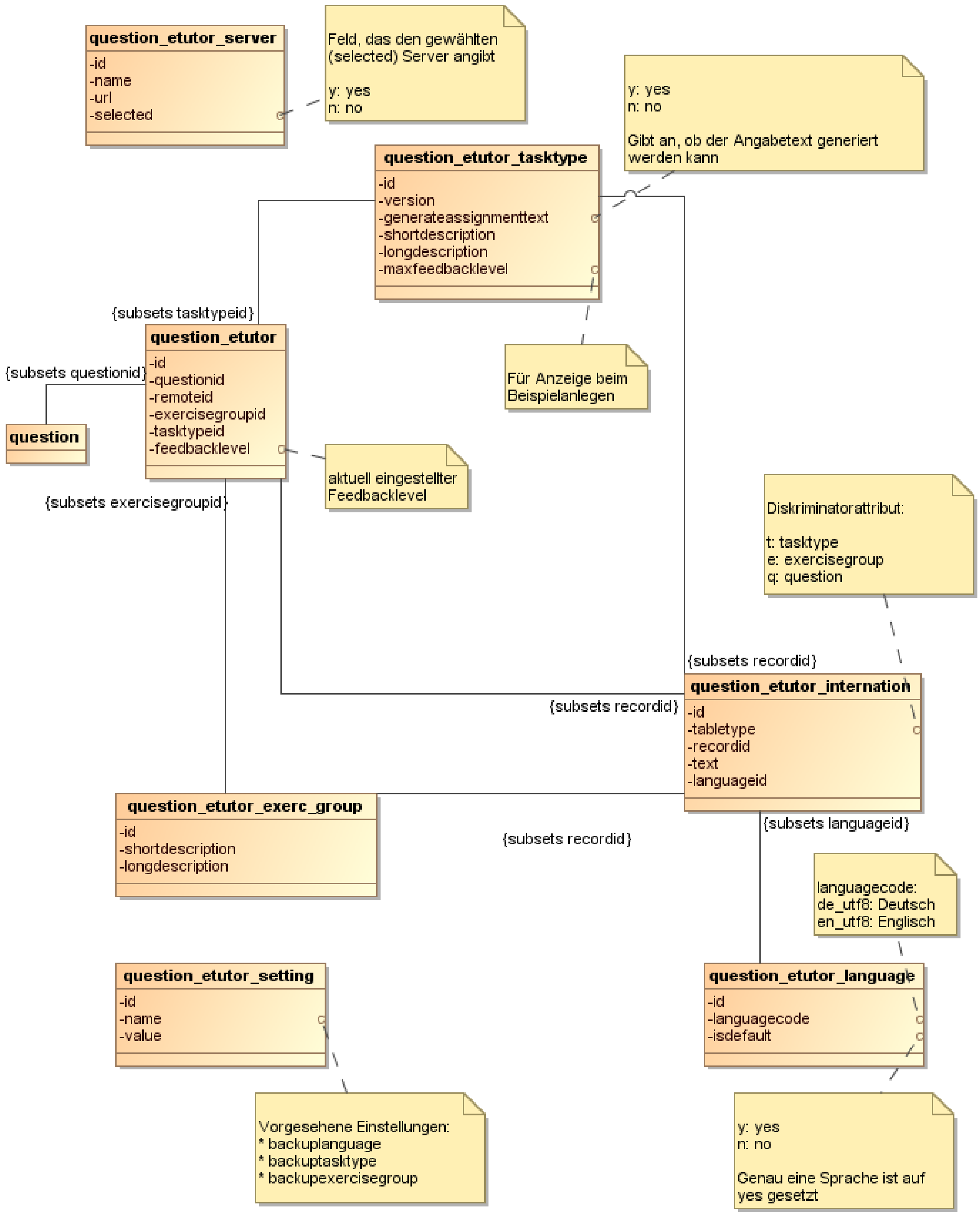


Abbildung 26 - Moodle-Datenbanktabellen zur Integration des eTutor-Beispieltyps

Das Moodle-Plugin *Tutoren-Bewertung* (Abbildung 27) wurde zur Unterstützung der manuellen Beispielkorrektur durch Tutoren entwickelt. Das Moodle-Kernsystem ermöglicht das Bereitstellen von Aufgabenstellungen (Tabelle *assignment*), und das Hochladen einer Lösung durch Studenten (*assignment_submissions*). Zusätzlich wurde die Tabelle *block_dke_tutor_assignment* zur Zuweisung von Übungsausarbeitungen an Tutoren erstellt. Die Tabelle *block_dke_tutor_uploadtask* erweitert Übungsaufgaben um eine Frist, bis zu der ein Tutor Zeit zur Korrektur hat.

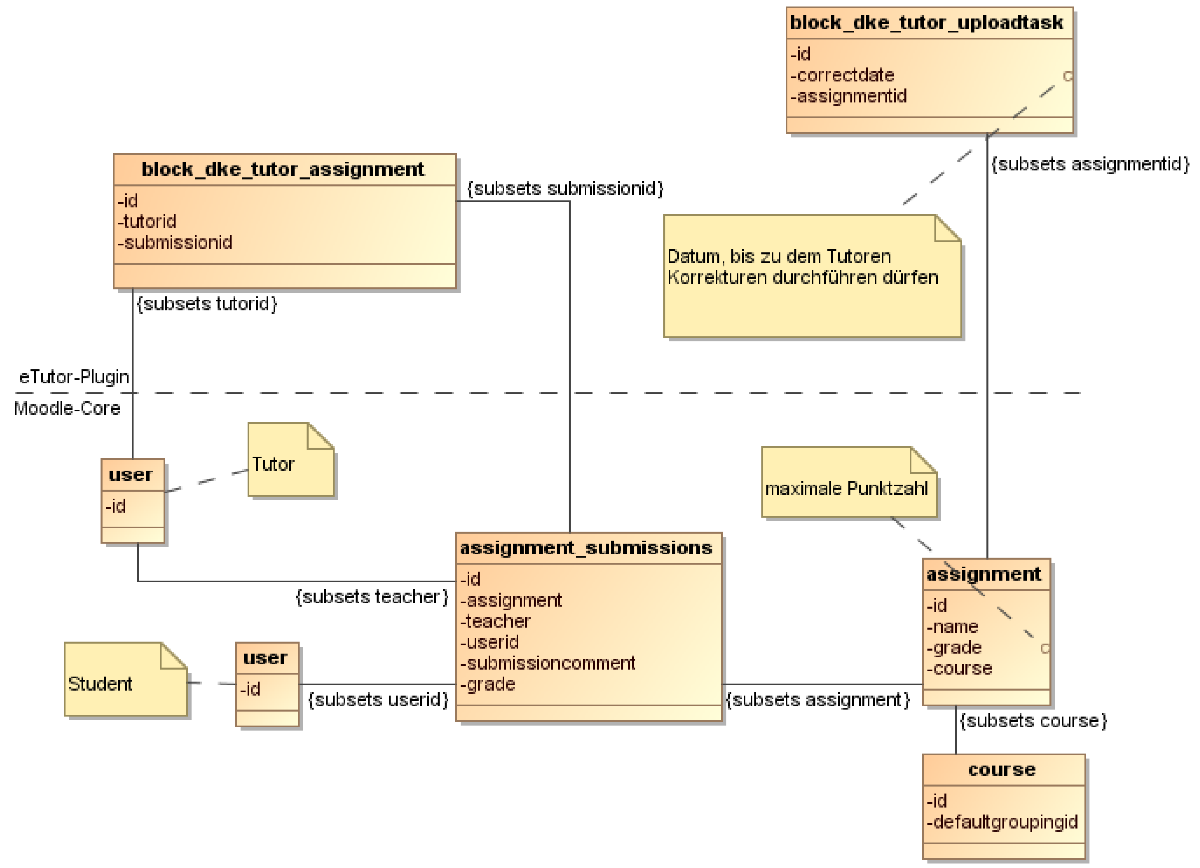


Abbildung 27 - Moodle-Datenbanktabellen für die manuelle Übungskorrektur durch Tutoren

4.2.4 Datenbank des Altsystems migrieren

Nachdem die Datenstrukturen des Zielsystems feststehen, kann die Zielsystem-Datenbank mit Inhalten gefüllt werden. Die Datenbankstruktur des eTutor-Altsystems ist in der Diplomarbeit von Anita Hofer beschrieben (Hofer 2005, S.177). Dabei ist es nicht erforderlich, Kursdaten vergangener Semester aus dem Altsystem zu übernehmen. Dazu gehören Informationen über Kursteilnehmer (euser[2]) und Übungszettel (taskassignment), die rein historische Zustandsdaten darstellen und im Anschluss an eine Archivierung verworfen werden können.

In den letzten Jahren wurde am DKE-Institut eine Vielzahl an Übungsbeispielen erstellt, die zur Vertiefung des vermittelten Lernstoffs genutzt wurden. Der aufwendig gepflegte Beispielbestand des eTutor-Altsystems sollte in die integrierte Ziel-Lernumgebung übernommen werden. Lösungen der Übungsbeispiele sind Teil der Expertenmodule, während Beispielinformationen wie Angabe und Übungsgruppe im eTutor-Core abgelegt waren. Die Expertenmodule wurden im Zuge der Migration beinahe unverändert gemeinsam mit den Beispiellösungen in das Zielsystem eingebaut. Da die eTutor-Core-Komponente nicht Teil des Zielsystems ist, mussten die verbleibenden Beispielinformationen in die Tabellenstruktur des Moodle-Systems migriert werden.

[2] Tabellenname im eTutor-Altsystem

Zur Übernahme von Beispielangaben in das Moodle-System ergibt sich folgende Vorgehensweise: Im ersten Schritt werden Daten zur Beispielkategorisierung wie die Übungsgruppe (*question_etutor_exerc_group*[3]) und der Beispieltyp (*question_etutor_tasktype*) benötigt. Bevor die textuellen Beispielangaben eingespielt werden können (*question_etutor_internation*), müssen alle Sprachen in die Datenbank eingetragen werden (*question_etutor_language*), in denen die Benutzerdialoge angezeigt werden können. Erst im letzten Schritt kann das eigentliche Beispiel angelegt werden (*question* und *question_etutor*).

In der folgenden Übersicht wird die Reihenfolge und das Mapping der einzuspielenden Daten angegeben. Dazu werden die Tabellen des eTutor-Altsystems und des Moodle-Zielsystems durch einen Folgepfeil getrennt angegeben.

1. Übungsgruppen migrieren: *exercisegroup* ⇨ *question_etutor_exerc_group*
2. Beispieltypen migrieren: *tasktype* ⇨ *question_etutor_tasktype*
3. unterstützte Sprachen für Angabetexte in *question_etutor_language* eintragen
4. Angabetexte übertragen und übersetzen: *exercisepool* ⇨ *question_etutor_internation*
5. Übungsbeispiel anlegen: *exercisepool* ⇨ *question, question_etutor*

4.3 Implementierung

Im Zuge der Migrationsbeschreibung in Kapitel 4.2.1.1 - Funktionalität wurden die Zielsystem-Komponenten in einer grafischen Übersicht (Abbildung 25) dargestellt. An dieser Stelle wird basierend auf dieser Grafik die Entwicklung der Zielsystem-Komponenten näher beschrieben. Für weiterführende Umgebungsbeschreibung wird auf die Anhänge Benutzerhandbuch (Anhang A), Installationshandbuch (Anhang B) und Administrationshandbuch (Anhang C) verwiesen.

4.3.1 Lernumgebung-Setup

Wie in Abbildung 28 dargestellt, ist die Zielsystem-Installation auf einen Moodle- und einen eTutor-Server aufgeteilt. *Moodle* kommt in Version *1.9* zum Einsatz, läuft auf einem *Apache 2*-Webserver mit PHP-Erweiterung und greift auf eine *MySQL*-Datenbank zu. eTutor setzt einen Java-Application-Server wie *Tomcat 6* voraus und verwendet die Java-Bibliotheken *Spring 2.5*, *Axis 2* und *Velocity 1.6*. Die meisten eTutor-Expertenmodule legen die Beispiel-Musterlösung in einer *Oracle*-Datenbank ab. Die Einrichtung dieser Komponenten wird im Installationshandbuch (Anhang B) näher beschrieben.

[3] Die in Klammer angeführten Moodle-Tabellen sind in Kapitel 4.2.3 näher beschrieben.

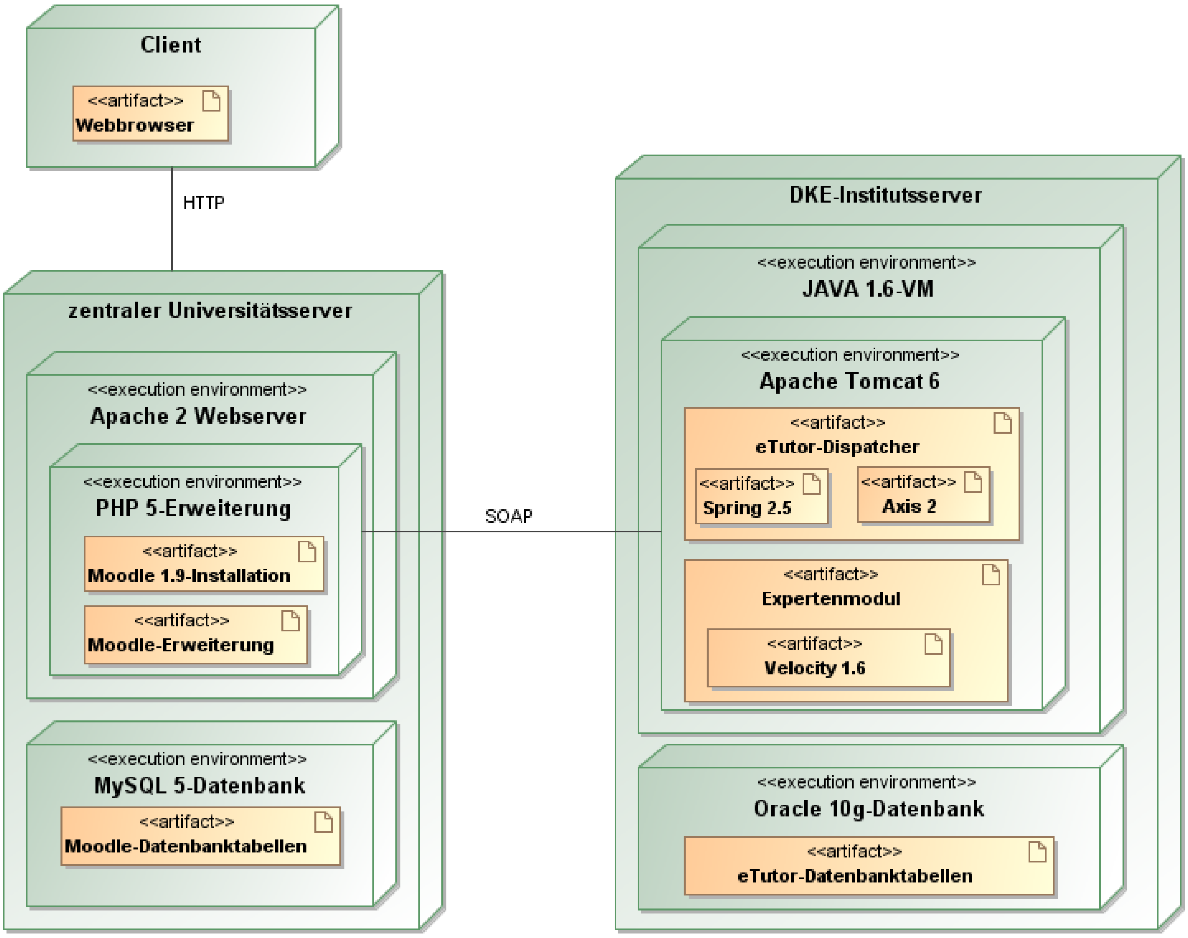


Abbildung 28 - Verteilungsdiagramm des integrierten Lernsystems

4.3.2 Entwicklungsumgebung

Das eTutor-Altsystem ist als Java-Projekt für die Entwicklungsumgebung *Eclipse* realisiert, und wird durch Ausführen eines Ant[4]-Skripts in eine Webanwendung exportiert. Im Zuge der Neuentwicklung des eTutor-Dispatchers wurde ein spezielles Webanwendungs-Projekt verwendet. Ein Vorteil davon ist, dass die Webanwendung von der Entwicklungsumgebung automatisch erstellt werden kann und dazu keine Skripte mehr erforderlich sind. Durch die Verwendung des in der Entwicklungsumgebung integrierten Webservers kann der Eclipse-Debugger zur Fehlersuche eingesetzt werden. Dadurch ist es beispielsweise möglich, die Variablenbelegung bei zuvor definierten Haltepunkten einzusehen.

Sowohl zur Erstellung der Moodle-Plugins als auch des eTutor-Dispatchers kam das Eclipse-Plugin *Web Tools Platform* zum Einsatz. Um die Moodle-Entwicklung zu unterstützen, wurde zudem der PHP-Debugger *XDebug* eingerichtet. Eclipse-Plugins für Java-Frameworks wie *Spring IDE* und *Veloeclipse* erleichterten die eTutor-seitige Implementierung des Zielsystems, da diese unter anderem nicht mit dem Java-Quelltext übereinstimmende Konfigurationsdateien erkennen können. Das Installationshandbuch (Anhang B) beschreibt die Konfiguration der Entwicklungsumgebung im

[4] http://ant.apache.org/

Detail. Im Rahmen der Diplomarbeit erstellter Programmcode wird in der DKE-institutsinternen Versionsverwaltung Subversion[5] verwaltet.

4.3.3 Kommunikation zwischen Moodle und eTutor

Im Folgenden wird zunächst die zur Kommunikation zwischen Moodle und eTutor eingesetzte Webservice-Technologie beschrieben. Daraufhin wird auf die Zwischenspeicherung von Dialog-Dateien näher eingegangen.

4.3.3.1 Webservice-Technologie

Webservice-Server beantworten Anfragen von Webservice-Clients und kommunizieren dabei im XML-Format. XML ist selbstbeschreibend, wodurch eine hohe Interoperabilität erreicht wird. Dadurch ist es möglich, dass Komponenten basierend auf unterschiedlichen Plattformen und Technologien miteinander kommunizieren. Ein Nachteil von XML ist sein Overhead, also der hohe Anteil an Steuerdaten im Bezug zur gesamten Nachricht. Die zusätzlichen Daten fallen bei der eTutor-Integration jedoch aufgrund des geringen Gesamt-Übertragungsvolumens während einer Übungsausarbeitung oder -erstellung kaum ins Gewicht.

Abbildung 29 stellt den Aufbau eines Webservice-Servers dar. Die *Listener*-Komponente reagiert auf Client-Anfragen und leitet diese an das *Business Interface* weiter, das nach Verarbeiten der XML-Struktur die darin enthaltenen Parameter an die *Business Logic* weiterreicht. Die Rückgabeparameter der *Business Logic* werden vom *Business Interface* wiederum in das XML-Format gebracht und vom *Listener* an den Client zurückgegeben.

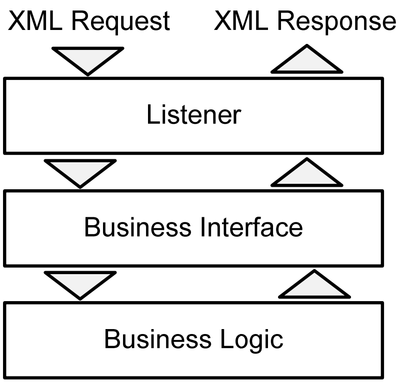


Abbildung 29 - Architektur eines Webservice in Anlehnung an (Vasudevan 2001)

Der Aufbau von Webservice-Nachrichten folgt dem SOAP-Standard. Kann eine Anfrage aufgrund eines Fehlers nicht korrekt beantwortet werden, schreibt SOAP ein bestimmtes Format vor, in dem die Fehlermeldung erfolgen muss; unter anderem ist die Angabe eines Fehlercodes obligatorisch. Die im eTutor-System verwendeten Fehlercodes finden sich im Administrationshandbuch (Anhang C).

[5] erreichbar unter der Adresse https://subversion.dke.uni-linz.ac.at/etutor

4.3.3.2 Zwischenspeichern von Dialog-Dateien

Beispieldialoge sind im HTML-Format realisiert, wobei ihr Aussehen je nach Beispieltyp stark variieren kann (vergleiche dazu Abbildung 30 und Abbildung 31). Diese können auf externe Ressourcen wie Bilder oder JavaScript-Dateien referenzieren. Da diese Dateien innerhalb des jeweiligen Expertenmoduls eTutor-seitig verwaltet werden, müssen sie mittels Webservice zu Moodle übertragen werden, bevor sie dem Benutzer angezeigt werden können (siehe Abbildung 24 - Kommunikationsablauf bei der Beispielbearbeitung).

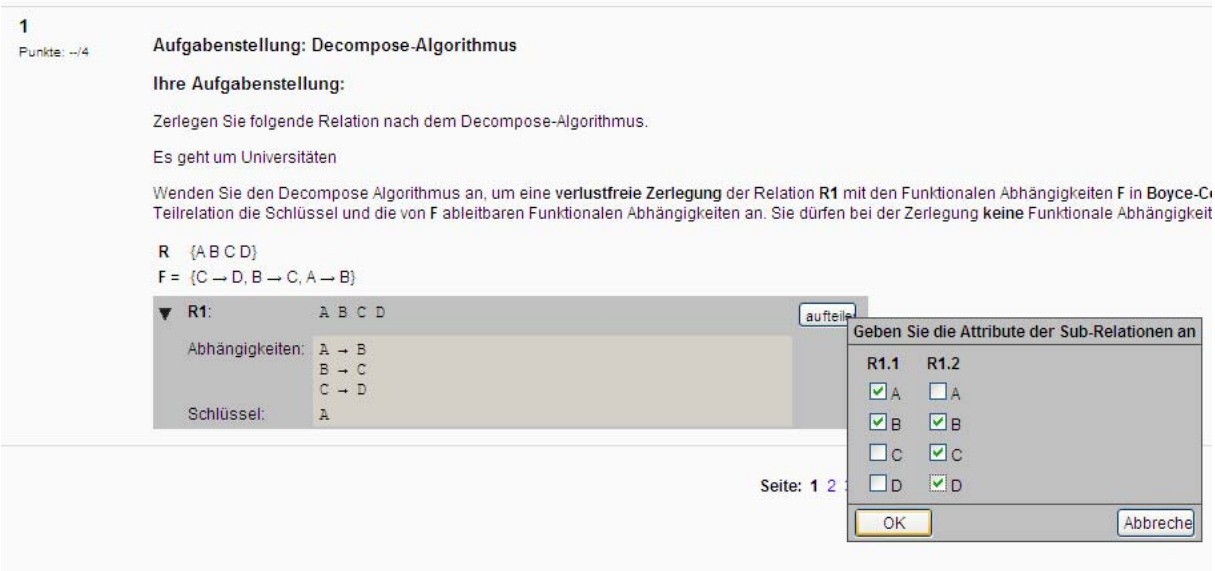


Abbildung 30 - Benutzerdialog zur Ausarbeitung einer Decompose-Übung

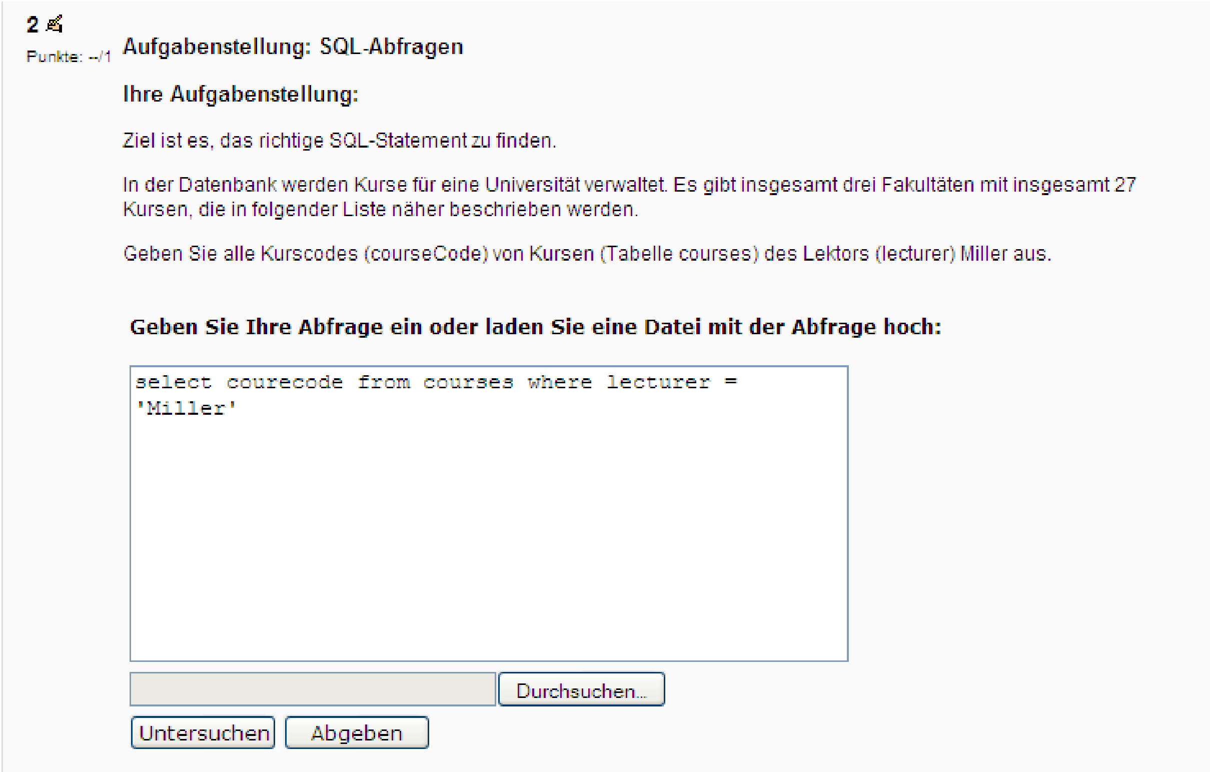


Abbildung 31 - Benutzerdialog zur Ausarbeitung einer SQL-Übung

Zur Reduzierung des Übertragungsvolumens sieht das implementierte Übertragungsprotokoll vor, dass nur jene Dateien übertragen werden, die sich noch nicht im Moodle-Zwischenspeicher befinden. Jede Änderung referenzierter Dialogdateien muss Moodle mitgeteilt werden, damit veraltete Dateien aus dem Zwischenspeicher entfernt werden können. Um die Gültigkeit von Dateien zu prüfen, wurde sowohl auf Moodle- als auch auf eTutor-Seite das Datenbankattribut *Version* eingeführt. Nur wenn beide Versionen eines Expertenmoduls übereinstimmen, sind die Dateien im Zwischenspeicher gültig. Soll nun beispielsweise ein Bild im SQL-Expertenmodul geändert werden, wird der *Version*-Parameter auf eTutor-Seite erhöht. Da sich dieses daraufhin vom Moodle-seitigen Version-Attribut unterscheidet, verlieren die zwischengespeicherten Dateien des SQL-Expertenmoduls ihre Gültigkeit und müssen erneut übertragen werden.

Im Folgenden soll die Zwischenspeicherung von Dateien anhand von drei Szenarien demonstriert werden. Szenario 1 zeigt, wie ein Zwischenspeicher aufgebaut wird. Dieser wird in Szenario 2 für den Zugriff auf eine benötigte Datei genutzt. Szenario 3 löst das Problem unterschiedlicher Dateiversionen und veranschaulicht die Aktualisierung zwischengespeicherter Dateien.

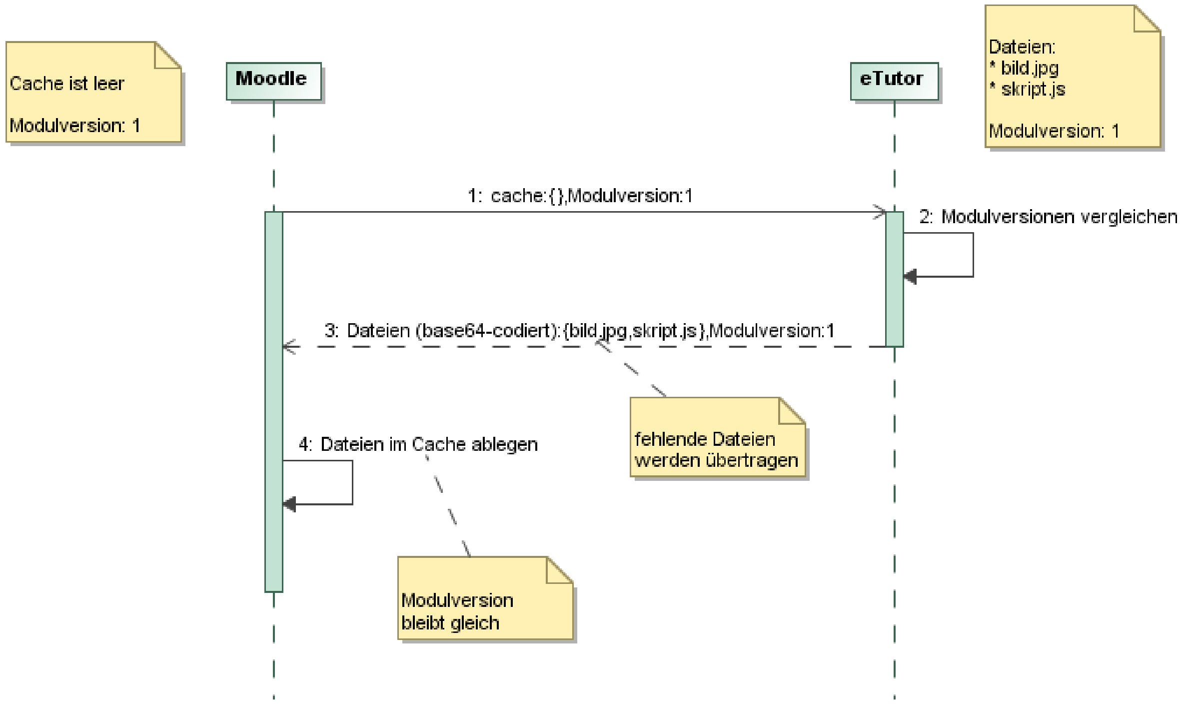


Abbildung 32 - Zwischenspeicher Szenario 1: keine Dateien vorhanden

In Abbildung 32 werden von einem Benutzerdialog die Dateien *bild.jpg* und *skript.js* referenziert. Da der Moodle-Zwischenspeicher zu Beginn keine Einträge enthält, werden alle vom Expertenmodul-Dialog referenzierten Dateien (Bilder, Skripten) mittels Webservice von eTutor zu Moodle übertragen und im Zwischenspeicher abgelegt.

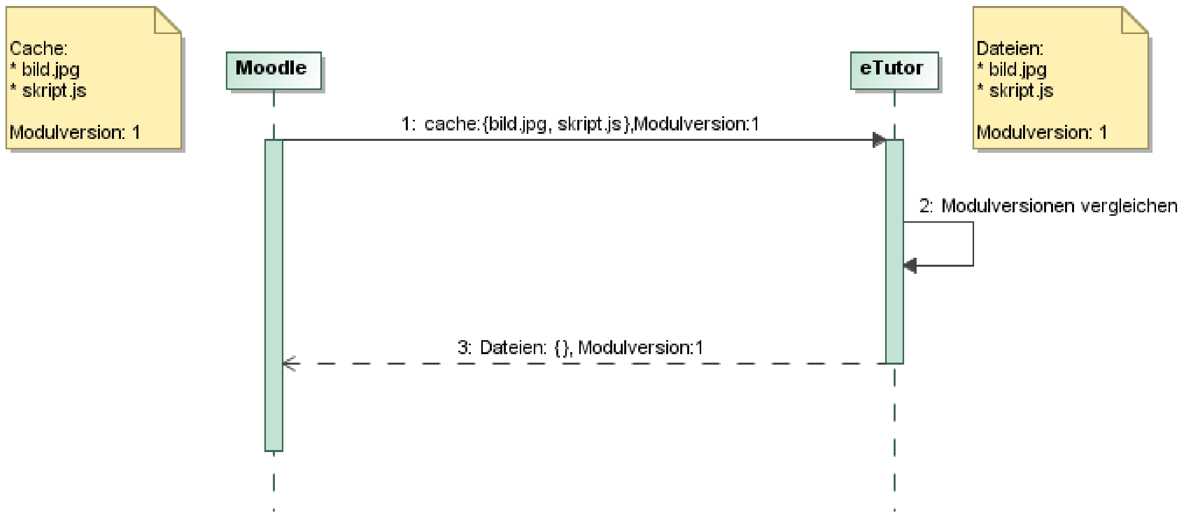


Abbildung 33 - Zwischenspeicher Szenario 2: nachgefragte Dateien in aktueller Version vorhanden

Abbildung 33 stellt den Idealfall dar, bei dem aufgrund des Moodle-Zwischenspeichers keine Datei-Übertragung notwendig ist. Moodle benötigt die Dateien *bild.jpg* und *skript.js*, die sich bereits im Zwischenspeicher befinden, und sendet zur Prüfung der Datei-Aktualität eine Webservice-Anfrage an

den eTutor. Da beide Versions-Parameter übereinstimmen, erkennt Moodle, dass die angeforderten Dateien nicht erneut übertragen werden müssen.

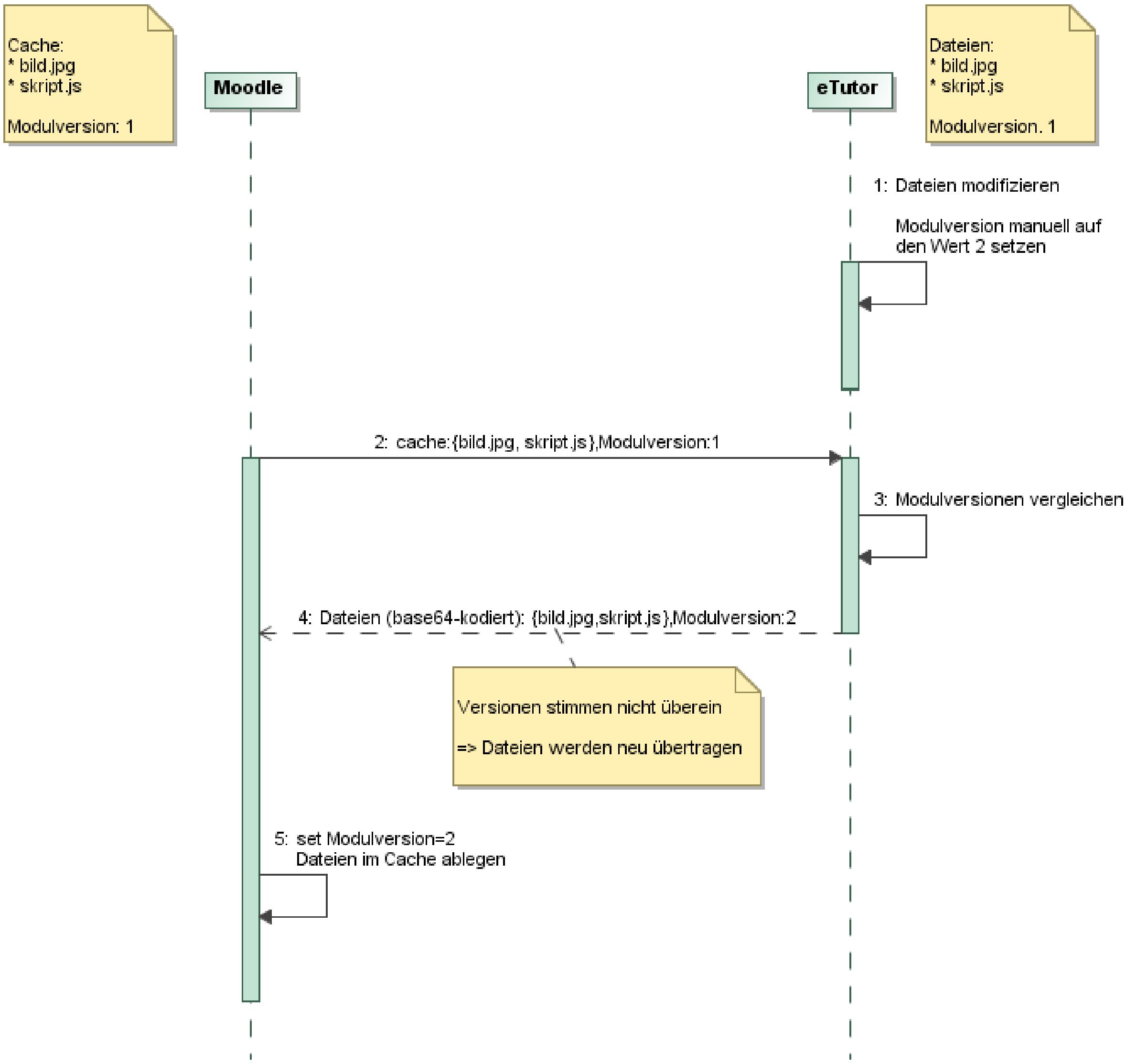


Abbildung 34 - Zwischenspeicher Szenario 3: veraltete Dateien vorhanden

Im dargestellten Szenario von Abbildung 34 modifiziert der Kursadministrator die Dateien *bild.jpg* und *skript.js* im eTutor-Expertenmodul. Um die Dateien als geändert zu markieren, muss der Administrator zudem die Modulversion manuell von Wert *1* auf *2* erhöhen. Das Setzen dieser Einstellung ist im Administrationshandbuch (Anhang C) beschrieben. Der Versions-Parameter wird dazu benötigt, dass veraltete Dateien im Moodle-Zwischenspeicher identifiziert und aktualisiert werden können und somit dem Lernenden immer die aktuelle Datei angezeigt wird. Mittels einer Moodle-Anfrage erkennt eTutor die abweichende Dateiversion und überträgt die geänderten Dateien erneut. Nachdem der Moodle-Zwischenspeicher auf den neuesten Stand gebracht wurde, wird vom Moodle-System das dort gespeicherte Versionsattribut ebenfalls auf den Wert *2* gesetzt. Bei einer

höheren Versionsnummer wird der gesamte Datenbestand des Expertenmoduls gelöscht und neu geladen.

4.3.4 Moodle-Komponenten

Moodle stellt eine Plugin-Schnittstelle zum Einbau neuer Funktionen bereit, die im Dokument *Introduction to Moodle-Programming* (Humboldt State University 2007) näher beschrieben wird. Derartige Erweiterungen haben auf die gesamte Moodle-Umgebung Zugriff (Datenbank-Tabellen, Session-Informationen, Benutzer und Rollen). Dabei ist es Aufgabe des Modul-Entwicklers, die Inhalte korrekt und entsprechend der Semantik von Moodle zu verändern, da Moodle keine Zugriffsrechte auf Modul-Ebene zur Verhinderung fehlerhafter Zugriffe implementiert.

Die Moodle-Erweiterung *Opaque* (Tim 2010) wurde von der Open University[6], einer Universität für Fernstudien, zur Einbindung von Übungsbeispielen ihres Lernsystems in Moodle entwickelt. Die Integration erfolgt mittels Webservice (Moodle Wiki 2010), wodurch beteiligte Komponenten lose gekoppelt werden können (siehe Beschreibung der Webservice-Technologie in Kapitel 4.3.3.1). Da Opaque unter der *GNU General Public License* veröffentlicht wurde, darf das Plugin ohne Einschränkung genutzt werden. Es ist daher erlaubt, die Funktionsweise der Erweiterung zu studieren und für eigene Zwecke anzupassen. Aus diesem Grund wurde die Einbindung des eTutor in Moodle in Anlehnung an Opaque realisiert.

Um Opaque für das Zielsystem nutzen zu können, mussten einige Korrekturen und Ergänzungen vorgenommen werden. Als Beispiel kann die veraltete Webservice-Schnittstellenbeschreibung genannt werden, die angepasst werden musste, um dem WS-I-Standard (WS-I Basic Profile 2006) zu entsprechen. WS-I spezifiziert den Aufbau einer WSDL-Datei, um Kompatibilität zwischen unterschiedlichen Webservice-Servern und Clients sicherzustellen. Darüber hinaus musste die Funktionalität des Übertragungsprotokolls dahingehend erweitert werden, dass Dateien nicht nur in Richtung Moodle sondern beidseitig übertragen werden konnten. Diese Funktionalität wird beispielsweise vom JDBC-Expertenmodul benötigt, bei dem Kursteilnehmer JAVA-Quelltexte hochladen, die innerhalb des JDBC-Expertenmoduls kompiliert und ausgeführt werden (Eichinger 2006, S.7).

4.3.5 eTutor-Komponenten

Um das Altsystem um eine Webservice-Schnittstelle zu erweitern, konnte entweder die existierende eTutor-Core-Webanwendung angepasst oder eine neue eTutor-Komponente mit Webservice-Funktionalität entwickelt werden. Da im Zielsystem ein großer Teil der Funktionalität des vorhandenen eTutors wegfällt (beispielsweise die Benutzerverwaltung), würde eine Modifikation des eTutor-Core des Altsystems zum Herauslösen nicht benötigter Funktionalitäten einen erheblichen Aufwand verursachen. Aus diesem Grund fiel die Entscheidung auf die Erstellung einer neuen

[6] http://www.open.ac.uk/

Komponente, den eTutor-Dispatcher. Dieser übernimmt Aspekte des Altsystems (für eine Übersicht der übernommenen Komponenten siehe Abbildung 21), verwendet jedoch eine vollständig neue Architektur und nutzt verschiedene Java-Bibliotheken (Abbildung 28). Diese Bibliotheken werden im Folgenden beschrieben. Außerdem werden Implementierungsalternativen genannt. Die Auswahl der Frameworks basierte jedoch auf keiner an bestimmten Kriterien festmachbaren Evaluierung, sondern fand nur aufgrund der Produktbeschreibungen statt.

4.3.5.1 Benutzeroberfläche

Die eTutor-Webanwendung des Altsystems setzt zur Darstellung der Benutzeroberfläche auf die Web-Technologien *Java-Servlet* und *JSP*. Da der eTutor-Dispatcher nicht als Webanwendung realisiert ist, sondern Benutzerdialoge via Webservice übertragen werden, müssen die Dialoge als Zeichenkette darstellbar sein. Dies war mit den Web-Technologien des Altsystems nur mit großem Aufwand zu erreichen, da JSP nicht unabhängig von der Servlet-Webanwendungs-Architektur betrieben werden kann. Für die Benutzeroberfläche des Zielsystems boten sich mehrere alternative Technologien wie *Velocity* (Velocity Dokumentation 2010), *XSLT* (XSLT-Dokumentation 1999) und *FreeMarker* (FreeMarker Dokumentation 2010) an. Wie innerhalb von JSP-Seiten können Velocity- und FreeMarker-Oberflächendefinitionen (auch Templates genannt) direkt auf Java-Klassen zugreifen; bei XSLT muss im Gegensatz dazu vor der Dialog-Anzeige die Oberflächendefinition zunächst in das XML-Format umgewandelt werden. Die Technologien *FreeMarker* und *Velocity* sind einander sehr ähnlich, jedoch existiert für Velocity eine größere Unterstützung durch Entwicklungswerkzeuge. Beispielsweise werden für die Entwicklungsumgebung *Eclipse* Erweiterungen zur Bearbeitung von Velocity-Templates angeboten. Velocity stellt somit die geradlinigste Umsetzung der JSP-Prinzipien dar und ermöglichte dadurch eine einfache Umstellung vorhandener JSP-Benutzerdialoge.

In Codeausschnitt 1 wird die bisher verwendete *JSP* (a)-Technologie der Templating-Bibliothek *Velocity* (b) des Zielsystems gegenübergestellt. In beiden Varianten wird die Methode *printReport* mit dem *Report*-Objekt als Übergabeparameter aufgerufen und der Rückgabewert im Benutzerdialog angezeigt.

<pre><%
Report report = null;
 	
if (session != null){
  report =
(Report)request.getAttribute(
RDBDConstants.PARAM_REPORT);
 	
  out.write(HTMLPrinter.printReport(
report, 0, 2, locale));
 	
}
%></pre>
	<pre>#if (${report})
	
	 $HTMLPrinter.printReport(${report},
	   0, 2, ${locale})
	
	#end</pre>

(a**) Jsp – printReport.jsp (Altsystem)**	**(b) Velocity Template - rdbdPrintReportView.vm**[7]

Codeausschnitt 1 - Gegenüberstellung: Benutzeroberfläche mit Jsp und Velocity

In der JSP-Variante werden die anzuzeigenden Daten aus der Benutzersession geladen und an der gleichen Stelle im Quelltext ausgegeben. Im Gegensatz dazu werden bei der Velocity-Umsetzung die anzuzeigenden Objekte zunächst, wie in Codeausschnitt 2 dargestellt, innerhalb des Java-Codes einer HashMap-Datenstruktur abgelegt und daraufhin dem entsprechenden Velocity-Template zur Anzeige übergeben. Bei Velocity wird daher die Datenaufbereitung von der Datenanzeige getrennt durchgeführt, wodurch sich die Übersichtlichkeit erhöht.

```
private String generateXhtml(Report report, Locale locale) {

  Map<String, Object> model = new HashMap<String, Object>();
  model.put("HTMLPrinter", HTMLPrinter.class);
  model.put("report", report);
  model.put("locale", locale);

  return VelocityEngineUtils.mergeTemplateIntoString(
    getVelocityEngine(),
    "resources/rdbd/printreportview/rdbdPrintReportView.vm", model
  ));
}
```

Codeausschnitt 2 - RDBDPrintReportView.java[8]

4.3.5.2 Komponenten-Integration

Um die vorhandenen Expertenmodule mit der neu entwickelten eTutor-Dispatcher-Komponente lose zu koppeln, kommt das Java-Framework *Spring* zum Einsatz. Spring basiert auf dem sogenannten *Inversion of Control* (IoC)- bzw. *Dependency Injection* (DI)-Ansatz (Spring-Dokumentation 2010). Dabei werden benötigte Ressourcen nicht direkt im Programmcode geladen, beispielsweise über einen JNDI-Lookup, sondern die Konfiguration der Abhängigkeiten findet außerhalb des Quelltexts statt.

---

[7] /implementation/etutor_dispatcher/src/resources/rdbd/printreportview/rdbdPrintReportView.vm
[8] /implementation/etutor_fd/src/etutor/modules/rdbd/ui/RDBDPrintReportView.java

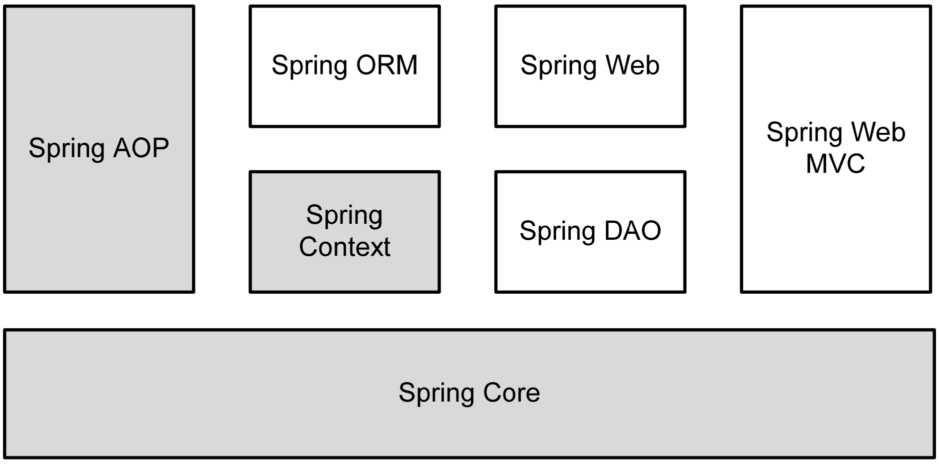


Abbildung 35 - Spring-Framework-Komponenten in Anlehnung an (Spring-Dokumentation 2010)

Das Spring-Framework ist eine Sammlung von Java-Bibliotheken, die zu mehreren Paketen zusammengefasst sind. Diese sind in Abbildung 35 dargestellt, wobei die grau hinterlegten Pakete mit im eTutor-Dispatcher-Projekt verwendet werden. *Spring Core* beinhaltet den sogenannten Container, der für die Verwaltung und die Kopplung von Anwendungskomponenten zuständig ist. Er stellt Anwendungskomponenten in Form von Plain Old Java Objects (POJOs) zur Verfügung, wobei unter POJOs klassische Java-Klassen ohne externe Abhängigkeiten verstanden werden (Martin 2010), die in unterschiedlichen Systemumgebungen lauffähig sind. Die Pakete *Spring Web* und *Spring Web MVC* enthalten Funktionen, die vor allem für Webanwendungen interessant sind. *Spring Context* ermöglicht Benutzerausgaben in unterschiedlichen Sprachen; dieser Mechanismus wird auch als Lokalisierung bezeichnet. *Spring DAO* und *Spring ORM* erlauben eine unkomplizierte Integration von objektrelationen Mappern und unterstützen Datenbank-Transaktionen und Exception-Handling. Mit *Spring AOP* wird sogenanntes Aspekt-orientiertes Programmieren ermöglicht. Damit können modulübergreifende Funktionalitäten an zentraler Stelle, dem sogenannten Aspekt, gekapselt und mehreren Modulen zugeordnet werden. Im eTutor-Dispatcher wird Aspekt-orientierte Programmierung für Logausgaben verwendet (siehe Kapitel 4.3.5.6 - Logging).

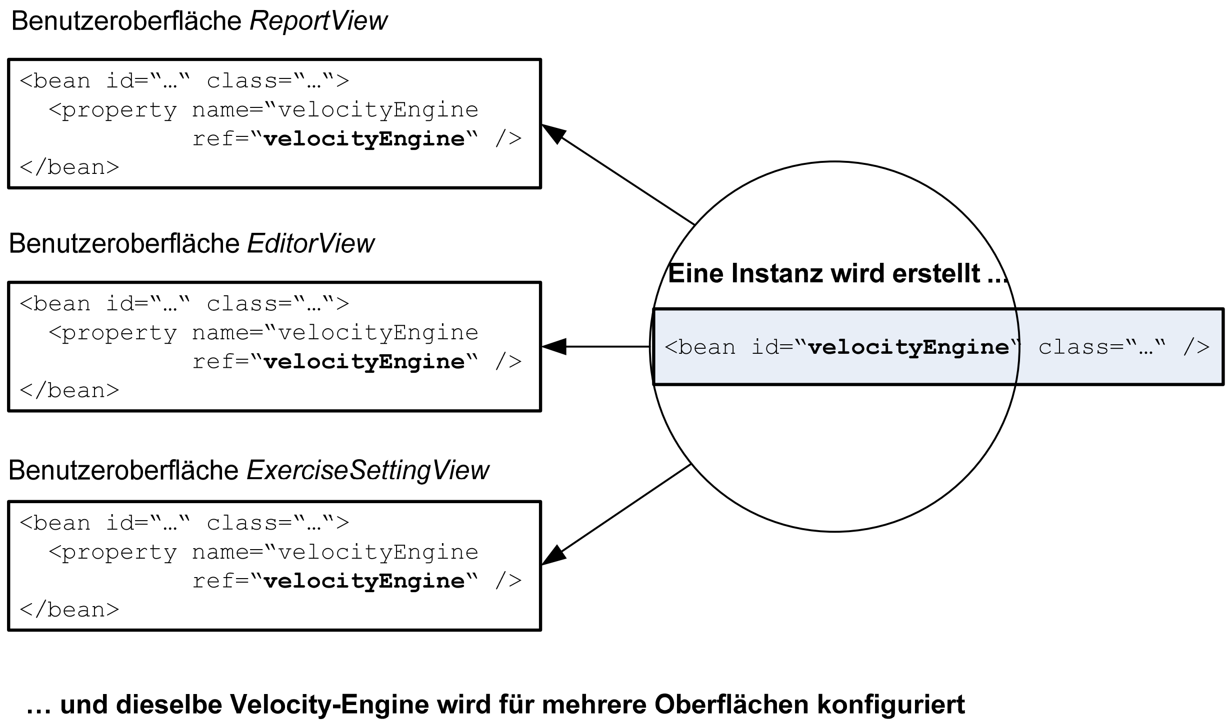


Abbildung 36 - Dependency Injection bei Spring

Das Spring-Framework unterstützt das Programmieren mithilfe von Interfaces. Anstatt konkrete Klassendefinitionen als Abhängigkeiten anzugeben, wird benötigte Funktionalität mittels Interfaces festgelegt. Dadurch kann eine referenzierte Klasse ohne Quelltext-Änderung durch eine andere Klasse ausgetauscht werden, vorausgesetzt diese implementiert das geforderte Interface. Die Definition von Abhängigkeiten mittels Interfaces und die lose Kopplung über Konfigurationsdateien ermöglicht das Bereitstellen klarer Schnittstellen für Lernmodule. Neue Expertenmodule, die diese Schnittstellen implementieren, können daraufhin ohne Quelltext-Modifikation nur durch Anpassung der Spring-Konfiguration in die eTutor-Dispatcher-Komponente eingebunden werden. Das Hinzufügen neuer Expertenmodule wird im Administrationshandbuch (Anhang C) beschrieben. Zudem erleichtert die Abhängigkeitsdefinition mittels Interfaces das isolierte Testen da in der Testumgebung referenzierte Komponenten durch Platzhalter, sogenannte Mock-Objekte, ausgetauscht werden können.

4.3.5.3 Webservice-Zugriff

Die Kommunikation zwischen eTutor-Dispatcher und Moodle wird mittels Webservice-Technologie realisiert, welche eine Datenübertragung im XML-Format gemäß dem SOAP-Standard vorsieht. Java-Frameworks wie *Axis1* und deren Nachfolger *Axis2* vereinfachen die Entwicklung von Webservice-Anwendungen insofern, als sich bei deren Einsatz Anwendungsentwickler nicht um den konkreten Aufbau der Webservice-Nachrichten kümmern müssen. Für den Einsatz von Axis1 würde sprechen, dass mit dem quelloffenen Lernsystem *OpenMark* bereits ein Java-System auf diese Schnittstelle setzt (OpenMark 2010) und somit eine Implementierungsvorlage existieren würde. Allerdings liegt die Veröffentlichung der letzten Axis1-Version bereits einige Jahre zurück und die Apache-Foundation,

unter deren Aufsicht Axis entwickelt wird (Apache Axis 2010), konzentriert sich auf die Verbesserung von Axis2, das laut Herstellerangaben wesentlich schneller sein soll als sein Vorgänger. Aus diesem Grund wurde im eTutor-Dispatcher Axis2 integriert.

Im Auslieferungsumfang von Axis2 ist das Programm *wsdl2java* enthalten. Dieses ermöglicht es, den Quelltext des Webservice-Servers ausgehend von der Schnittstellendefinition automatisch zu erstellen. Diese Vorgehensweise entspricht dem *Contract-First*-Prinzip. Demgegenüber würde das *Contract-Last*-Prinzip vorsehen, zuerst einen Webservice-Server zu entwickeln und auf diesen abgestimmt eine Schnittstellenbeschreibung zu erstellen (Smith u. a. 2005). Bei der Entwicklung des eTutor-Core wurde gemäß Contract-First vorgegangen, wobei die genaue Vorgehensweise im Administrationshandbuch (Anhang C) beschrieben ist.

4.3.5.4 Sessionverwaltung

Das Bearbeiten eines Übungsbeispiels erfordert in den meisten Fällen mehrere Dialog-Interaktionen. Dabei ist es erforderlich, den aktuellen Bearbeitungsstatus einem Benutzer zuordnen zu können. Innerhalb der Moodle-Webanwendung wird der aktuelle Zustand in einer Benutzersession verwaltet. Da bei verteilten Systemen mit Webservice-Kommunikation kein System-übergreifender Zustand existiert, musste, wie in Codeausschnitt 3 dargestellt, die Verwaltung von Sitzungsinformation eTutor-seitig nachgebildet werden.

```
public interface SessionManager {
  public Session getSession(String key) throws InvalidSessionException;

  public void removeSession(String key) throws InvalidSessionException;

  public int getSessionCount();

 //lockSessionObjekt wird zur Vermeidung konkurrierender Zugriffe benötigt
  public static Object lockSessionObject = new Object();
}
```

Codeausschnitt 3 - SessionManager[9]-Klasse zur Verwaltung von Zustandsinformationen

Damit mehrere Lernende zur gleichen Zeit Übungsbeispiele ausarbeiten können, muss der SessionManager Zugriffe von mehreren parallelen Threads koordinieren. Dazu wird beim Erstellen des SessionManagers ein *lockSessionObject* angelegt. Das Problem konkurrierender Zugriffe wird dadurch gelöst, dass immer nur ein Thread exklusiv auf das lock-Objekt zugreifen kann (Oaks & Wong 2004). Ein Thread kann daher nur dann die Anweisungen der in Codeausschnitt 4 mit dem *synchronized*-Befehl begrenzten Bereiche ausführen, wenn dieser zu diesem Zeitpunkt Zugriff auf das lock-Objekt hat.

[9] /implementation/etutor_dispatcher/src/etutor/core/ws/session/SessionManager.java

```
  public Session getSession(String key) throws InvalidSessionException {
    Session session = null;

    synchronized(lockSessionObject) {
      session = sessions.get(key);
      if (session == null) {
        session = new Session();
        sessions.put(key, session);
      }
      long timestampNow = (new Date()).getTime();
      session.setTimestamp(timestampNow);
    }

    return session;
  }

  public void removeSession(String key) throws InvalidSessionException {

    synchronized(lockSessionObject) {
      sessions.remove(key);
    }

  }
…
```

Codeausschnitt 4 - SessionManagerImpl.java[10]

4.3.5.5 Mehrsprachigkeit

Sowohl Moodle als auch der eTutor-Dispatcher unterstützen mehrsprachige Benutzerdialoge. Dazu befinden sich im Programmcode anstatt konkreter Texte Verweise auf Konfigurationsdateien, welche Benutzerausgaben an zentraler Stelle verwalten. Wie in Codeausschnitt 5 dargestellt, können diese Konfigurationen für mehrere Sprachen angelegt werden. Im Rahmen der Diplomarbeit wurde die gesamte Benutzeroberfläche Deutsch- (b) und Englisch-sprachig (a) zur Verfügung gestellt.

… decomposereporter.first=First decomposereporter.second=Second decomposereporter.third=Third …	… decomposereporter.first=Erste decomposereporter.second=Zweite decomposereporter.third=Dritte …
(a**) messages_en.properties**[11]	**(b) messages_de.properties**[12]

Codeausschnitt 5 - Definition mehrsprachiger Ausgabetexte

Das Hinzufügen neuer Sprachen und die Modifikation vorhandener Sprachtexte werden im Administrationshandbuch (Anhang C) beschrieben.

[10] /implementation/etutor_dispatcher/src/etutor/core/ws/session/impl/SessionManagerImpl.java
[11] /implementation/etutor_dispatcher/src/messages_en.properties
[12] /implementation/etutor_dispatcher/src/messages_de.properties

4.3.5.6 Logging

Im laufenden Betrieb können Textausgaben Hinweise über den aktuellen Status der Programmausführung geben. Der eTutor-Dispatcher protokolliert daher alle Funktionsaufrufe, um den Programmablauf nachvollziehen zu können.

Zur Ausgabe von Statusmeldungen könnte zu Beginn jeder Funktion deren Funktionsname ausgegeben werden (siehe Codeausschnitt 6 (a)). Problematisch an dieser Lösung ist der große Aufwand zum Eintragen der Logausgabe an vielen verschiedenen Stellen. Das gleiche Problem tritt beim Anpassen der Logausgaben auf, wenn etwa zusätzlich die Übergabeparameter einer Funktion ausgegeben werden sollen.

```
package
etutor.core.ws.session.impl;

public class SessionManagerImpl
implements SessionManager {

  public Session getSession(String
key) {

    if (logger.isDebugEnabled())
      logger.debug("getSession(
        String) - start");
    ...

    if (logger.isDebugEnabled())
      logger.debug("getSession(
        String) - end");
  }
}
```

(a) Beispiel für Logausgaben direkt im Funktionscode

```
@Pointcut("within(etutor..*),
within(test.etutor..*)")
protected void getPublic() {
}

@Around("getPublic()")
public Object doBasicProfiling(
  ProceedingJoinPoint pjp) {
  String methodname =
    pjp.getSignature().getName();

  logger.debug(methodname
    + " arguments: " +
    Arrays.toString(pjp.getArgs()));
  Object returnValue = pjp.proceed();

  logger.debug(methodname
    + " returns " + returnValue);
  return returnValue;
}
```

(b) AOP: zentraler ActivityLoggingAspect.java[13]

Codeausschnitt 6 - Logausgaben im Funktionscode versus Aspect-oriented Programming (AOP)

Im eTutor-Dispatcher-Projekt wurde daher eine Lösung mittels Aspekt-orientierter Programmierung gewählt. Dabei sind wiederkehrende Kommandos zu Beginn oder am Ende einer Funktion an zentraler Stelle, dem sogenannten Aspekt (Codeausschnitt 6 (b)), gespeichert. Eine weiterführende Erklärung zur Konfiguration des Aspekts befindet sich im Administrationshandbuch (Anhang C).

4.3.6 Weitere Funktionalitäten

In diesem Abschnitt wurde ausführlich auf die Entwicklung des eTutor-Dispatchers und dessen Kommunikation mit Moodle eingegangen. Um die geforderte Zielsystemfunktionalität (siehe Auflistung in Abbildung 21) vollständig zu implementieren, wurde außerdem jeweils ein Moodle-

[13] /implementation/etutor_dispatcher/src/test/etutor/core/ws/ActivityLoggingAspect.java

Plugin zum Administrieren der Beispieldatenbank und zur Beispiel-Korrektur durch Tutoren entwickelt. Deren Funktionsweise ist im Benutzerhandbuch (Anhang A) näher erklärt.

4.3.7 Beurteilung

Die Integration der Lernsysteme eTutor und Moodle wurde mittels Webservices gelöst. Dank klar definierter Schnittstellen kann die Kommunikation zwischen den Lernsystemen nachvollzogen werden. Änderungen am Moodle-Kernsystem waren für die Systemumstellung nicht notwendig. Die in Kapitel 4.2.1.4 - Zielsystem-Funktionalität definierten technischen Vorgaben zur Updatefähigkeit von Moodle konnten durch die ausschließliche Verwendung vorgegebener Erweiterungsschnittstellen weitgehend erfüllt werden. Lediglich für den Fall, dass sich in einer zukünftigen Moodle-Version der Aufbau der Erweiterungsschnittstelle ändern sollte, könnte im Zuge einer Lernsystem-Aktualisierung eine Anpassung von Moodle-Plugins erforderlich sein.

Das erstellte Lernsystem kann an mehreren Stellen erweitert werden. Im Zielsystem müssen zum Einbinden neuer Korrekturfunktionalität nur Konfigurationsdateien und keine Quelltexte geändert werden. Nach dem gleichen Prinzip können lokalisierte Ausgabetexte in Benutzerdialogen um weitere Sprachen ergänzt werden. Übungsbeispiele könnten zudem nicht nur für bewertete Übungsbeispiele, sondern auch für freiwillige Übungen zur Wiederholung des Lernstoffs verwendet werden.

Das Lernsystem könnte auch um eine Lernfortschrittskontrolle erweitert werden (Teufl 2008). Dabei würde die Auswahl der Übungsbeispiele dem Kenntnisstand eines Kursteilnehmers angepasst werden. Da alle Bewertungen im Moodle-System verwaltet werden, würde sich eine Implementierung mittels Moodle-Erweiterung anbieten. Eine vereinfachte Variante wäre durch ein eigenständiges System realisierbar, das in die Webservice-Kommunikation der Übungsausarbeitung eingebunden werden würde. Dabei könnte die Detailliertheit des Feedbacks von der Anzahl der protokollierten Abgabeversuche des Studenten abhängig gemacht werden. Somit würden erst dann detailliertere Lösungshinweise angezeigt, wenn sich Kursteilnehmer länger mit einer Aufgabe beschäftigt haben.

5 Zusammenfassung

Das intelligente tutorielle System *eTutor* wurde am Institut für Data and Knowledge Engineering an der Johannes Kepler Universität entwickelt und ermöglicht die automatische Korrektur von Übungsbeispielen. Zur Verwaltung von Universitätskursen wird an der Johannes Kepler Universität das Learning Management System *Moodle* angeboten.

Vorrangiges Ziel der Diplomarbeit war die Integration vorhandener Korrekturfunktionalität des eTutors in das Learning Management System *Moodle*. Im Zuge einer Migration sollte das eigenständige eTutor-System (Altsystem) auf eine kombinierte eTutor und Moodle Umgebung (Zielsystem) umgestellt werden. Grund dafür war die nur rudimentär umgesetzte Verwaltungs-Funktionalität des eTutor-Altsystems, welche durch Moodle ersetzt werden sollte. Die Korrekturfunktionalität des eTutor-Systems, die sich in sogenannten Expertenmodulen befindet, erfüllt alle Anforderungen und sollte daher so weit wie möglich beibehalten werden. Es wurde darauf geachtet, Komponenten des resultierenden Lernsystems lose zu koppeln. So waren zum Einbinden von Expertenmodulen nur Adaptionen der Benutzerdialoge notwendig, die restlichen Funktionen zur automatischen Aufgabenkorrektur konnten ohne Änderung des Quelltexts übernommen werden.

Im Zuge der Integration mussten einige Rahmenbedingungen eingehalten werden. Am Moodle-Kernsystem durften keine Anpassungen vorgenommen werden, da diese andernfalls vor jeder Aktualisierung von Moodle in die neue Programmversion eingebaut werden müssten. Moodle bietet zur Umgehung dieses Problems eine Erweiterungsschnittstelle, deren Struktur auch bei neuen Programmversionen beibehalten wird. Da sowohl für den Einbau neuer Funktionalität als auch für die Integration des eTutors ausschließlich die dafür vorgesehenen Schnittstellen genutzt wurden, kam es zu keiner Einschränkung der Updatefähigkeit von Moodle. Eine weitere Anforderung war, vorhandene Beispieldaten in die neu erstellte Lernumgebung übernehmen zu können. Da die Korrekturfunktionalität und damit auch die Speicherung der Musterlösung in den Expertenmodulen vollständig beibehalten wurde, mussten lediglich die Angabetexte und Daten zur Kategorisierung von Übungen in das Zielsystem eingespielt werden.

Über die Basisanforderungen hinaus wurden neue Funktionen wie mehrsprachige Benutzeroberflächen umgesetzt. Übungsbeispiele können in der erstellten Lernumgebung sowohl auf Deutsch als auch auf Englisch ausgearbeitet und administriert werden. Dies erleichtert vor allem nicht deutschsprachigen Austauschstudenten die Teilnahme an Lehrveranstaltungen. Die Verwaltung der Texte erfolgt mittels Konfigurationsdateien, weshalb neue Sprachen ohne Quelltext-Änderung flexibel hinzugefügt werden können. Ähnlich funktioniert die Einbindung von Expertenmodulen in den eTutor. Zum Hinzufügen neuer Expertenmodule genügt es, eine Konfigurationsdatei im eTutor-Projekt anzupassen. Zudem wurde darauf geachtet, die Einbindung des eTutors in Moodle mittels Webservices derart allgemein zu

halten, dass die eTutor-Korrekturfunktionalität prinzipiell auch mit anderen LMS als *Moodle* verwendet werden kann.

Erweiterungspotential bietet vor allem die ITS-Architektur. Derzeit implementiert eTutor die Funktionalität eines Expertenmoduls und Moodle dient als Kommunikationsmodul. Moodle erfasst zwar Bewertungen von Kursteilnehmern, nutzt diese Information jedoch nicht, um Kursinhalte individualisiert zur Verfügung zu stellen. Ein idealtypisches ITS würde für die Erfassung des Lernfortschritts ein Studentenmodul und zur Auswahl adäquater Kursinhalte wie Übungsbeispiele und Lernskripten ein gekapseltes Unterrichtsmodul vorsehen. Derartige Module könnten auch für die Moodle-eTutor-Lernumgebung entwickelt und mittels Webservice-Schnittstelle in das bestehende Lernsystem eingebunden werden.

Literaturverzeichnis

Apache Axis, 2010. Apache Web Services Project. Available at: http://ws.apache.org/ [Zugegriffen April 5, 2010].

Arnold, J.T., 2007. Learning on the Fly. *HRMagazine*, 52(9), 127-131.

Bing, W. u. a., 1997. Legacy systems migration-a method and its tool-kit framework. In *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*. Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference. Hong Kong, S. 312-320.

Bisbal, J. u. a., 1999. Legacy Information Systems: Issues and Directions. *IEEE Software*, 16, 103-111.

Blackboard, 2010. Blackboard Learn Platform. Available at: http://www.blackboard.com/Teaching-Learning/Learn-Platform.aspx [Zugegriffen Juni 12, 2010].

Brodie, M. & Stonebraker, M., 1995. *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach: Gateways, Interfaces and the Incremental Approach*, Morgan Kaufmann Publishers In.

Brooks, C. u. a., 2006. Combining ITS and eLearning Technologies: Opportunities and Challenges. In *Intelligent Tutoring Systems*. S. 278-287.

Brusilovsky, P., 2003. A component-based distributed architecture for adaptive webbased education. In *Artificial intelligence in education*. IOS Press.

Eichinger, C., 2006. Moduldokumentation JDBC.

Faeskorn-Woyke, H. u. a., 2009. Tools für die Lehre im Fach Datenbanken. *Datenbank-Spektrum*, 9(29), 5–13.

FreeMarker Dokumentation, 2010. FreeMarker Manual. Available at: http://freemarker.sourceforge.net/docs/index.html [Zugegriffen April 3, 2010].

Garrison, R. & Kanuka, H., 2004. Blended learning: Uncovering its transformative potential in higher education. *The Internet and Higher Education*, 7(2), 95-105.

Hofer, A., 2005. *E-Exercises in Data & Knowledge Engineering - Konzeption und Entwicklung eines intelligenten tutoriellen Systems*. Johannes Kepler Universität.

Humboldt State University, 2007. Kurs: Introduction to Moodle Programming. Available at: http://dev.moodle.org/course/view.php?id=2 [Zugegriffen Juni 16, 2010].

Juric, M. u. a., 2000. Integrating legacy systems in distributed object architecture. *SIGSOFT Softw. Eng. Notes*, 25(2), 35-39.

Koenings, B., 2010. *UML-Tutor - Konzeption und Entwicklung eines halbautomatisierten Systems zur Bewertung und Beurteilung von konzeptuellen Datenbankschemata im*

Rahmen des intelligenten tutoriellen Systems eTutor. Johannes Kepler Universität.

Lusti, M., 1992. *Intelligente tutorielle Systeme : Einführung in wissensbasierte Lernsysteme.*, München , Wien : Oldenbourg,

Martin, F., 2010. MF Bliki: POJO. Available at: http://www.martinfowler.com/bliki/POJO.html [Zugegriffen Mai 19, 2010].

Moodle, 2010. Moodle.org: open-source community-based tools for learning. Available at: http://moodle.org/ [Zugegriffen März 31, 2010].

Moodle Statistics, 2010. Moodle.org: Moodle Statistics. Available at: http://moodle.org/stats/ [Zugegriffen April 15, 2010].

Moodle Wiki, 2010. Open protocol for accessing question engines. Available at: http://docs.moodle.org/en/Development:Opaque [Zugegriffen Mai 14, 2010].

Oaks, S. & Wong, H., 2004. *Java Threads* 3. Aufl., O'Reilly Media.

OpenMark, 2010. OpenMark-Dokumentation. Available at: https://openmark.dev.java.net/nonav/javadoc/ [Zugegriffen Mai 5, 2010].

Papazoglou, M. & Georgakopoulos, D. hrsg., 2003. Service-oriented computing. *Communications of the ACM*.

Rabin, J. & McCathieNevile, C., 2008. Mobile Web Best Practices 1.0. *Mobile Web Best Practices*. Available at: http://www.w3.org/TR/mobile-bp/#d0e1321 [Zugegriffen April 5, 2010].

Sakai, 2010. Sakai Project - an Open Source suite of learning, portfolio, library and project tools | Sakai Project. Available at: http://sakaiproject.org/ [Zugegriffen März 31, 2010].

Schneidewind, N., 1998. How to evaluate legacy system maintenance. *IEEE Software*, 15(4), 34-42.

Smith, E. u. a., 2005. Rethinking the Java SOAP Stack. *IEEE International Conference on Web Services (ICWS)*, 12-15.

Spring-Dokumentation, 2010. Spring Reference. Available at: http://static.springsource.org/spring/docs/2.5.x/reference/introduction.html [Zugegriffen April 18, 2010].

Stevens, P. & Pooley, R., 1998. Systems reengineering patterns. In *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '98/FSE-6*. the 6th ACM SIGSOFT international symposium. Lake Buena Vista, Florida, United States, S. 17-23. Available at: http://portal.acm.org/citation.cfm?doid=288195.288210 [Zugegriffen April 4, 2010].

Teufl, M., 2008. *Individualisierte Ablaufsteuerung in einem Intelligenten Tutoriellen System - Konzeption und Umsetzung am Beispiel des ITS „eTutor“*. Johannes Kepler

Universität.

Tim, H., 2010. Moodle.org: Modules and plugins. Available at: http://moodle.org/mod/data/view.php?d=13&rid=798 [Zugegriffen Juni 16, 2010].

Vasudevan, V., 2001. A Web Services Primer. *webservices.xml.com*. Available at: http://www.xml.com/pub/a/ws/2001/04/04/webservices/ [Zugegriffen April 18, 2010].

Velocity Dokumentation, 2010. Apache Velocity - Velocity User Guide. Available at: http://velocity.apache.org/engine/devel/user-guide.html [Zugegriffen April 3, 2010].

WS-I Basic Profile, 2006. Basic Profile - Version 1.1 (Final). Available at: http://www.ws-i.org/Profiles/BasicProfile-1.1.html [Zugegriffen April 15, 2010].

XSLT-Dokumentation, 1999. XSL Transformations (XSLT). Available at: http://www.w3.org/TR/xslt [Zugegriffen April 3, 2010].

Zhang, D. u. a., 2004. Can e-learning replace classroom learning? *Commun. ACM*, 47(5), 75-79.

Anhang

A. Benutzerhandbuch

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeausschnittverzeichnis

1 Überblick

Der Hauptbildschirm von Moodle ist nach dem Login dreigeteilt. In der Mitte werden Kursinhalte angezeigt, wie eine Übersichtsliste über Lehrmaterialen eines Kurses oder der Inhalt eines Lehrmaterials, während sich links und rechts davon sogenannte Kursblöcke befinden. Dabei handelt es sich um Zusatzinformationen zum Kurs, wie Informationen über neueste Diskussionsforumsbeiträge oder eine Liste von festgelegten Weblinks. Die Reihenfolge der Darstellung von Kursblöcken kann angepasst werden.

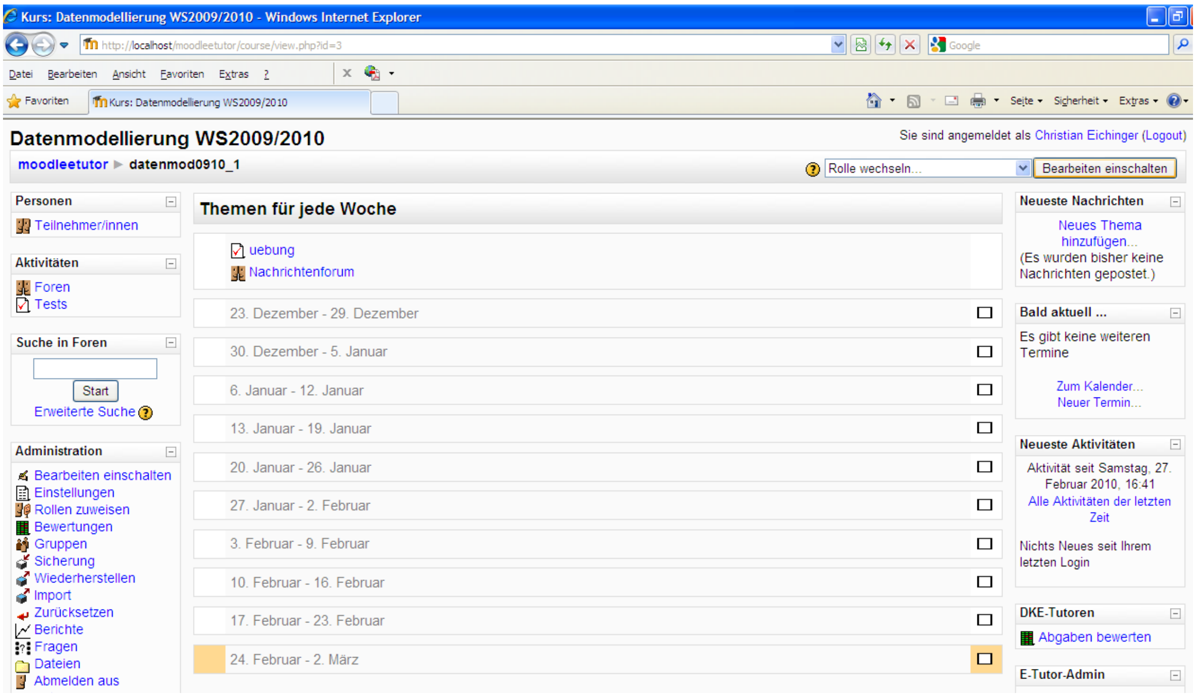


Abbildung 1 - Der Moodle-Hauptbildschirm ist geteilt in Kursinhalte und Kursblöcke

Interaktionen mit Kursteilnehmern, die über das Betrachten von Lehrmaterialien hinausgehen, werden unter dem Begriff Lernaktivität zusammengefasst. Beispiele für Lernaktivitäten sind Formulare zum Hochladen von Übungsausarbeitungen oder Fragebögen mit Multiple-Choice-Fragen zur Überprüfung des vermittelten Kursinhalts.

Die Kursansicht stellt die Übersichtsseite eines Kurses dar. In dieser Ansicht werden alle Arbeitsmaterialen und Lernaktivitäten eines bestimmten Kurses aufgelistet. Diese Kursinhalte können unterschiedlich angeordnet werden, beispielsweise nach Wochen oder Themengebieten sortiert.

Im Rahmen der Diplomarbeit wurden die zwei Kursblöcke *DKE-Tutoren* und *E-Tutor-Admin* entwickelt. Sind Funktionen direkt über einen Kursblock verlinkt, wird die entsprechende Verknüpfung zu Beginn der Funktionsbeschreibung angeführt.

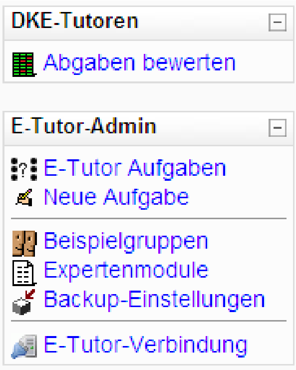


Abbildung 2 - neu erstellte Kursblöcke

2 Beispieldatenbank administrieren (Kursleiter)

Änderungen an der Beispieldatenbank dürfen nur von Kursleitern und Systemadministratoren durchgeführt werden.

2.1 Übungsbeispiel erstellen

Kursblock *E-Tutor-Admin*: Neue Aufgabe

Zum Erstellen einer Übungsaufgabe werden Daten in mehreren Schritten erhoben. Der erste Dialog enthält Einstellungen, die für Beispiele jedes Beispieltyps gleich sind, während die Gestaltung der folgenden Dialoge je nach Beispieltyp variiert.

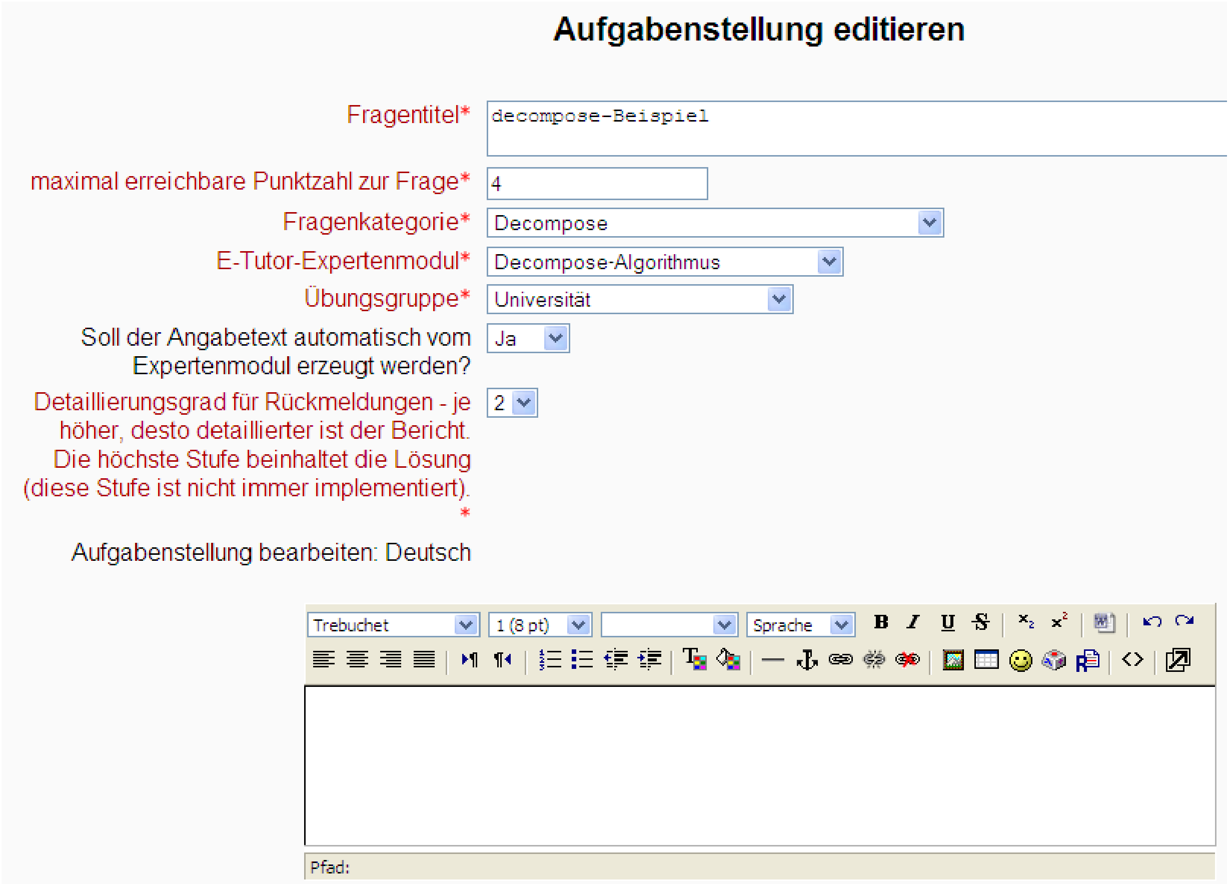


Abbildung 3 - beispieltypübergreifende Einstellungen

Für jedes Übungsbeispiel müssen ein Titel und die zu erreichende Punktzahl angegeben werden. Dabei sollte beachtet werden, dass die maximale Punktzahl nur eine Vorgabe darstellt, die beim Anlegen eines Tests abgeändert werden kann. Mittels Auswahllisten werden die Fragenkategorie, der Beispieltyp (bestimmt das zu verwendende eTutor-Expertenmodul) und die Übungsgruppe festgelegt. Übungsgruppen kategorisieren Beispiele in Themenbereiche (siehe dazu auch Abschnitt 2.1.1). Damit kann die Themenbeschreibung zentral abgelegt werden (siehe Kapitel 2.3). Dadurch erspart man sich, bei jeder textuellen Beispielangabe erneut eine Einführung in den behandelten Themenbereich zu geben. Bei manchen Beispieltypen unterstützt das entsprechende eTutor-Expertenmodul ein automatisches Generieren des Angabetexts. In diesem Fall braucht das Eingabefeld *Aufgabenstellung bearbeiten* nicht ausgefüllt werden. Wenn das Expertenmodul eine automatische Angabetext-Erstellung nicht unterstützt, ist die Auswahlbox deaktiviert und die Option zur automatischen Aufgabenerstellung kann in diesem Fall nicht ausgewählt werden.

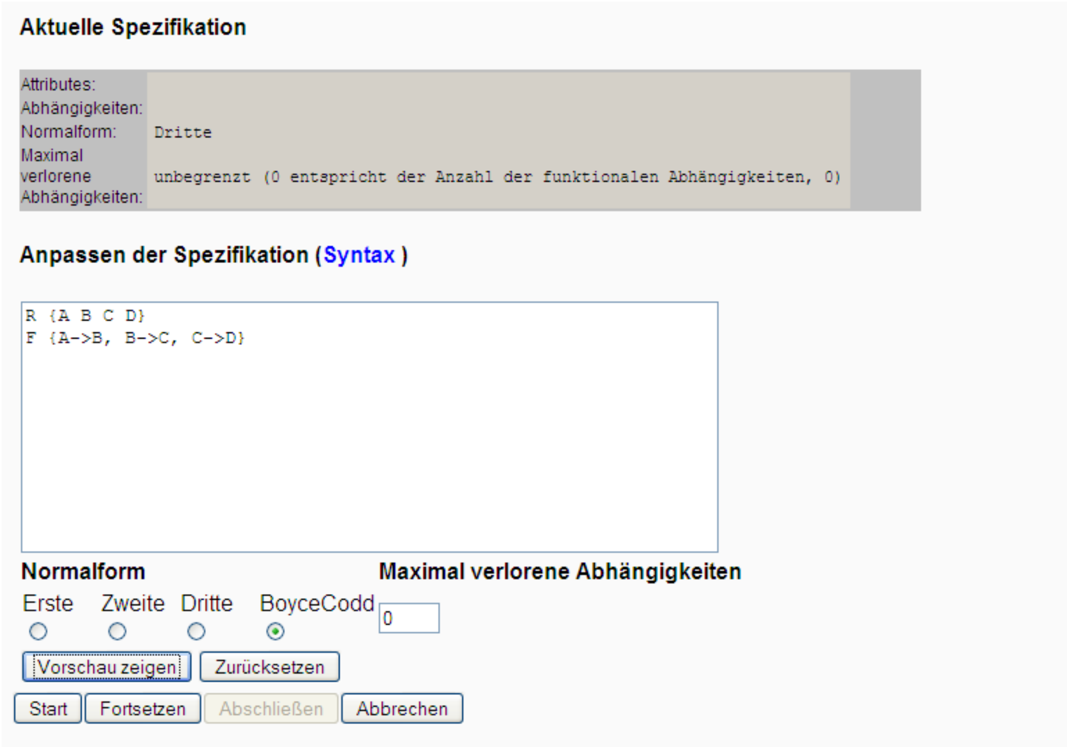


Abbildung 4 - Erstellung einer Übung vom Beispieltyp Decompose

Abbildung 4 zeigt, wie eine Übungsaufgabe zum Decompose-Verfahren angelegt wird. Mit dem Decompose-Algorithmus können Relationen in eine höhere Normalform gebracht werden. Neben der Beschreibung der im Beispiel verwendeten Relation und deren Attribut-Abhängigkeiten muss die gewünschte Ziel-Normalform gewählt werden. Außerdem wird die Anzahl der maximal zu verlierenden Abhängigkeiten angegeben.

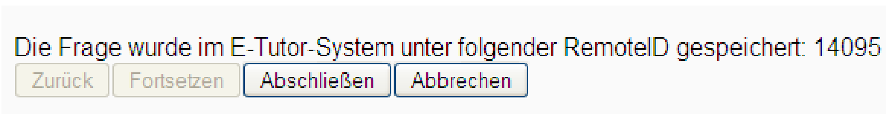


Abbildung 5 - Bestätigung zum Ablegen in der Beispieldatenbank

Erst wenn die korrekte Speicherung der Beispiel-Musterlösung von eTutor-Seite bestätigt wurde, kann das Übungsbeispiel über die Schaltfläche *Abschließen* in der Moodle-Datenbank abgelegt werden. Diese Maßnahme stellt bei der verteilten Transaktion sicher, dass jedes in Moodle abgelegte Übungsbeispiel erfolgreich im eTutor-System gespeichert wurde. Das bedeutet, dass nur jene Beispiele verwendet werden können, die auch tatsächlich funktionsfähig sind.

2.1.1 Beispielgruppe erstellen/editieren

Kursblock *E-Tutor-Admin*: Beispielgruppen

Mehrere Übungsbeispiele des selben Themenbereichs können in Beispielgruppen zusammengefasst werden. Eine Beispielgruppe enthält eine textuelle Einführung des jeweiligen Themas; diese Themenbeschreibung wird über der Angabe jedes Beispiels angezeigt, das sich in dieser Gruppe

befindet. Beispielsweise existieren mehrere Übungsaufgaben zum Thema Universitätskurse. Das Konzept der Beispielgruppen ermöglicht es, die Beschreibung dieser Kurse an zentraler Stelle in der Beispielgruppe zu verwalten; in der Übungsbeispiel-Angabe muss nur die tatsächliche textuelle Aufgabenstellung gespeichert werden.

Das Anlegen und Editieren einer Beispielgruppe kann mit dem Vorgehen beim Verwalten von Übungsbeispielen verglichen werden. In der Beispielgruppen-Übersicht können im Lernsystem verwaltete Beispielgruppen geändert oder gelöscht werden; der Dialog zum Erstellen neuer Beispielgruppen ist über die Verlinkung *Neue Beispielgruppe erstellen* erreichbar.

2.1.2 Expertenmodul-Bezeichnung anpassen

Kursblock *E-Tutor-Admin*: Expertenmodule

Das Einbinden zusätzlicher Expertenmodule wird im Administrationshandbuch (Anhang C) beschrieben.

2.2 Übungsbeispiele auflisten

Kursblock *E-Tutor-Admin*: E-Tutor Aufgaben

Die aufgelisteten Übungsbeispiele können nach verschiedenen Kriterien gefiltert werden. Nach dem Aufruf des Beispieldialogs sind keine Einschränkungen aktiv. Mittels Auswahlboxen können nur jene Beispiele angezeigt werden, die zu einem bestimmten Expertenmodul oder einer Übungsgruppe gehören. Idealerweise sind für jedes Übungsbeispiel die Angabetexte in allen unterstützten Sprachen verfügbar. Um nun all jene Beispiele mit fehlendem englischsprachigen Angabetext anzuzeigen, muss die Option *fehlender Angabetext in Sprache* auf *Englisch* gesetzt werden.

Neben jeder Übungsaufgabe sind Verknüpfungen zum Editieren und Löschen von Übungsbeispielen angeführt.

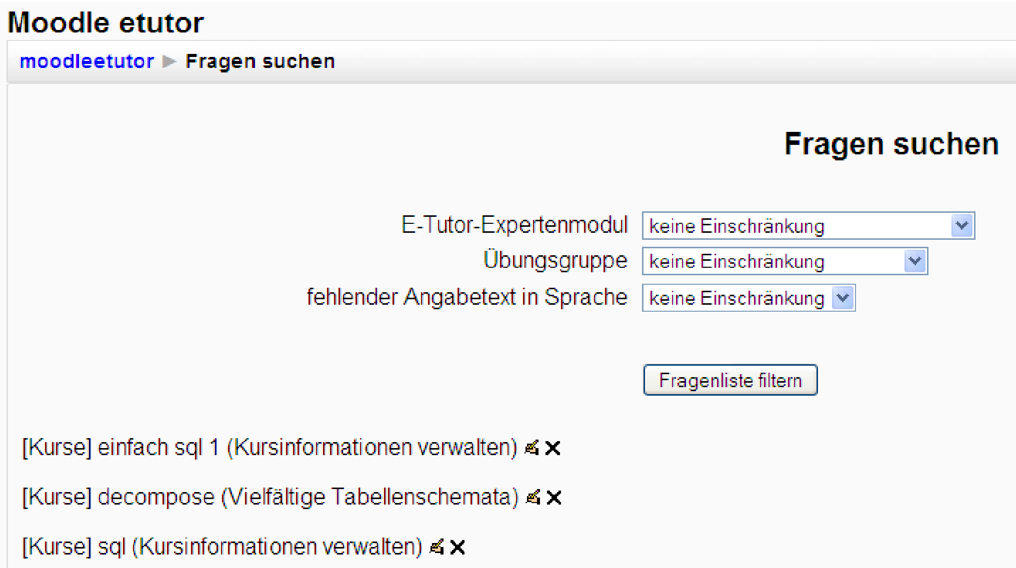


Abbildung 6 - Übersichtsdialog zur Verwaltung des Beispielpools

2.3 Beispielgruppen verwalten

Kursblock *E-Tutor-Admin*: Beispielgruppen

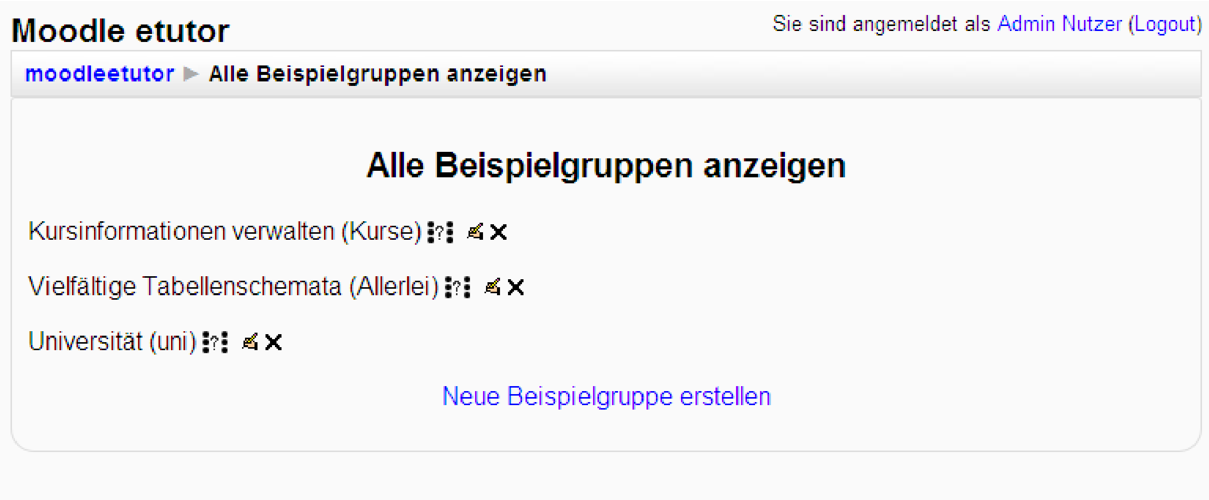


Abbildung 7 - Übersichtsdialog zur Verwaltung von Übungsgruppen

In einem Übersichtsdialog werden alle im System verwalteten Beispielgruppen angezeigt. Beispielgruppen können über entsprechende Verknüpfungen neben dem Beispielgruppen-Namen geändert oder gelöscht werden. Unter der Liste befindet sich eine Option zum Hinzufügen einer neuen Beispielgruppe. Im Gegensatz zur Übungsbeispiel-Übersicht können in diesem Dialog die angezeigten Beispielgruppen nicht nach Kriterien gefiltert werden, da voraussichtlich nur wenige Beispielgruppen im System verwaltet werden.

2.4 Kurs- und Übungsbeispiel-Backup

Abbildung 8 – Schaltfläche "Sicherung" im Administrationsblock zum Kurs-Backup

Der Kursblock *Administration* ermöglicht über die Funktion *Sicherung* das Backup von Kursen[14]. Beim Erstellen einer Kurs-Sicherung wird festgelegt, ob Kursteilnehmer und Lernmaterialien wie Übungszettel mitgespeichert werden sollen. Über den Kursblock *E-Tutor-Admin* (siehe Abbildung 2) kann zudem mithilfe des Menüpunkts *Backup-Einstellungen* eingestellt werden, ob bei einem Backup von Übungsaufgaben abhängige Daten wie Beispielgruppen mit gesichert werden sollen. In Codeausschnitt 1 ist das ausschließliche Sichern von Übungsbeispielen (a) dem Speichern aller beispielabhängigen Informationen (b) gegenübergestellt. Die Auswahl letzterer Option hat den Vorteil, dass beim Einspielen von Beispieldaten nicht auf Abhängigkeiten geachtet werden muss. Nachteilig ist die Tatsache, dass Zusatzinformationen wie Beispielgruppen redundant zu jedem Übungsbeispiel gespeichert werden und dadurch die Menge der Sicherungsdaten vergleichsweise groß ist.

[14] http://docs.moodle.org/en/Course_backup

```
<QUESTION>
  <ID>30</ID>
  <PARENT>0</PARENT>
  <NAME>[uni] ra-frage</NAME>
  ...
  <ETUTOR>
    <REMOTEID>13113</REMOTEID>
    <FEEDBACKLEVEL>3</FEEDBACKLEVEL>
    <TASKTYPESHORTDESCRIPTION>
      ra
    </TASKTYPESHORTDESCRIPTION>
    <EXERCISEGROUPSHORTDESCRIPTION>
      uni
    </EXERCISEGROUPSHORTDESCRIPTION>
```

```
<QUESTION>
  <ID>30</ID>
  <PARENT>0</PARENT>
  <NAME>[uni] ra-frage</NAME>
  ...
  <ETUTOR>
    <REMOTEID>13113</REMOTEID>
    <FEEDBACKLEVEL>3</FEEDBACKLEVEL>
    <TASKTYPE>
      <SHORTDESCRIPTION>
        ra
      </SHORTDESCRIPTION>
      <LONGDESCRIPTION>
        relationale Algebra
      </LONGDESCRIPTION>
      <VERSION>5</VERSION>
      <GENERATEASSIGNMENTTEXT>
        n
      </GENERATEASSIGNMENTTEXT>
      <MAXFEEDBACKLEVEL>
        4
      </MAXFEEDBACKLEVEL>
      <INTERNATIONS>
        <INTERNATION>
          <TEXT>Ziel ist es, den
richtigen relationalen Algebra-
Ausdruck zu finden.</TEXT>
          <LANGUAGE>
            <LANGUAGECODE>
              de_utf8
            </LANGUAGECODE>
            <ISDEFAULT>
              y
            </ISDEFAULT>
          </LANGUAGE>
        </INTERNATION>
        <INTERNATION>
          <TEXT>Create the correct
relational Algebra term.</TEXT>
          <LANGUAGE>
            <LANGUAGECODE>
              en_utf8
            </LANGUAGECODE>
            <ISDEFAULT>
              n
            </ISDEFAULT>
          </LANGUAGE>
        </INTERNATION>
      </INTERNATIONS>
    </TASKTYPE>
    <EXERCISEGROUP>
      <SHORTDESCRIPTION>
        uni
      </SHORTDESCRIPTION>
      <LONGDESCRIPTION>
        Universität
      </LONGDESCRIPTION>
      <INTERNATIONS>
        <INTERNATION>
          <TEXT> Es geht um
Universitäten </TEXT>
```

```
    <INTERNATIONS>
      <INTERNATION>
        <TEXT>return coursecode
(tablecourse) from lecturer
'Miller'</TEXT>
           <LANGUAGECODE>en_utf8
           </LANGUAGECODE>
      </INTERNATION>
      <INTERNATION>
        <TEXT>Coursecode (Tabelle
course) von lecturer 'Miller'
ausgeben</TEXT>
        <LANGUAGECODE>de_utf8
        </LANGUAGECODE>
      </INTERNATION>
    </INTERNATIONS>
  </ETUTOR>
</QUESTION>
```

(a) reiner Export von Fragedaten

```
          <LANGUAGE>
            <LANGUAGECODE>
              de_utf8
            </LANGUAGECODE>
            <ISDEFAULT>
              y
            </ISDEFAULT>
          </LANGUAGE>
        </INTERNATION>
        <INTERNATION>
          <TEXT> It's all about
universities </TEXT>
          <LANGUAGE>
            <LANGUAGECODE>
              en_utf8
            </LANGUAGECODE>
            <ISDEFAULT>
              n
            </ISDEFAULT>
          </LANGUAGE>
        </INTERNATION>
      </INTERNATIONS>
    </EXERCISEGROUP>
    <INTERNATIONS>
      <INTERNATION>
        <TEXT>return coursecode
(tablecourse) from lecturer
'Miller'</TEXT>
        <LANGUAGE>
          <LANGUAGECODE>
            en_utf8
          </LANGUAGECODE>
          <ISDEFAULT>
            n
          </ISDEFAULT>
        </LANGUAGE>
      </INTERNATION>
      <INTERNATION>
        <TEXT>Coursecode (Tabelle
course) von lecturer 'Miller'
ausgeben</TEXT>
        <LANGUAGE>
          <LANGUAGECODE>
            de_utf8
          </LANGUAGECODE>
          <ISDEFAULT>
            y
          </ISDEFAULT>
        </LANGUAGE>
      </INTERNATION>
    </INTERNATIONS>
  </ETUTOR>
</QUESTION>
```

(b) Export aller abhängiger Daten

Codeausschnitt 1 - Backup von Aufgabenstellungen

Kursblock *Administration*: Wiederherstellung

Das Wiederherstellen von Kurselementen wird im Moodle-Benutzerhandbuch[15] beschrieben.

Hinweis	Beim Einspielen bereits im System vorhandener Übungsbeispiele werden Duplikate angelegt.

3 Kurs administrieren (Kursleiter)

Die Administration des Kurses fällt in den Zuständigkeitsbereich des Kursleiters.

3.1 Kurs anlegen

Das Anlegen eines Kurses ist für Administratoren unter dem Menüpunkt *Kurse/Verwaltung* möglich. Kurse können in sogenannten Kursbereichen kategorisiert und somit beispielsweise Instituten zugeordnet werden. Im nächsten Schritt können über den Administrations-Block (siehe Abbildung 8) mittels der Funktion *Rollen zuweisen* Studenten in den Kurs eingeschrieben werden. Zudem kann der Administrator den Lehrveranstaltungsleitern die (Kurs)-Trainer-Rolle zuweisen. Diese Kurs-Trainer haben daraufhin die Möglichkeit, Tutoren (Rolle: *dke-Tutor*) zur manuellen Beispiel-Korrektur zu ernennen.

Im nächsten Schritt können Administrationsblöcke hinzugefügt werden. Dazu werden innerhalb des Kurshauptmenüs mit der Schaltfläche *Bearbeiten einschalten – Blöcke (Hinzufügen)* die Blöcke *DKE-Tutoren* und *E-Tutor-Admin* ausgewählt.

3.2 Übungszettel erstellen

Moodle-Kernfunktionalität wie das Anlegen eines Übungszettels werden detailliert in der Moodle-Benutzeranleitung[16] beschrieben. In den folgenden Unterpunkten wird auf die Besonderheit beim Erstellen von eTutor-Übungsbeispielen eingegangen.

3.2.1 Automatisch korrigierte Übungsaufgabe bereitstellen

Um neue Testfragen zu erstellen, sollte der Dialog *Neue Aufgabe* des Blocks *E-Tutor-Admin* verwendet werden. Wichtig ist die Auswahl des korrekten E-Tutor-Expertenmoduls, da abhängig von diesen unterschiedliche Eingabemasken für den Beispielinhalt angezeigt werden.

Um die Testfragen nutzen zu können, wird die Aktivität *Test* (über *Bearbeiten einschalten – Aktivität anlegen – Test*) hinzugefügt. Im Erstellungsdialog müssen die Einstellungen *Zahl der Fragen pro Seite* auf *1* und *Max. Zahl der Versuche* auf *1* gesetzt werden. Diese Einstellungsänderung ist deswegen notwendig, da ansonsten aus einem Beispielpool zugeteilte Übungsbeispiele bei jedem Testversuch erneut zufällig zugewiesen werden würden. Das eTutor-System lässt trotz dieser Einstellung eine beliebige Anzahl an Versuchen zu. Im Reiter *Bearbeiten* können aus der Fragenliste Fragen

[15] http://docs.moodle.org/en/Course_backup
[16] http://docs.moodle.org/en/Adding/updating_a_quiz

ausgewählt und mithilfe des Doppelpfeil-Icons einem Test zugewiesen werden. Dabei sollte man auf die Auswahl der korrekten Fragenkategorie achten, da nur die Fragen der aktuell ausgewählten Kategorie aufgelistet werden. Die gewählten Übungsbeispiele können im Reiter *Vorschau* vom Trainer- oder Admin-Nutzer ausprobiert und von den Kursteilnehmern bearbeitet werden.

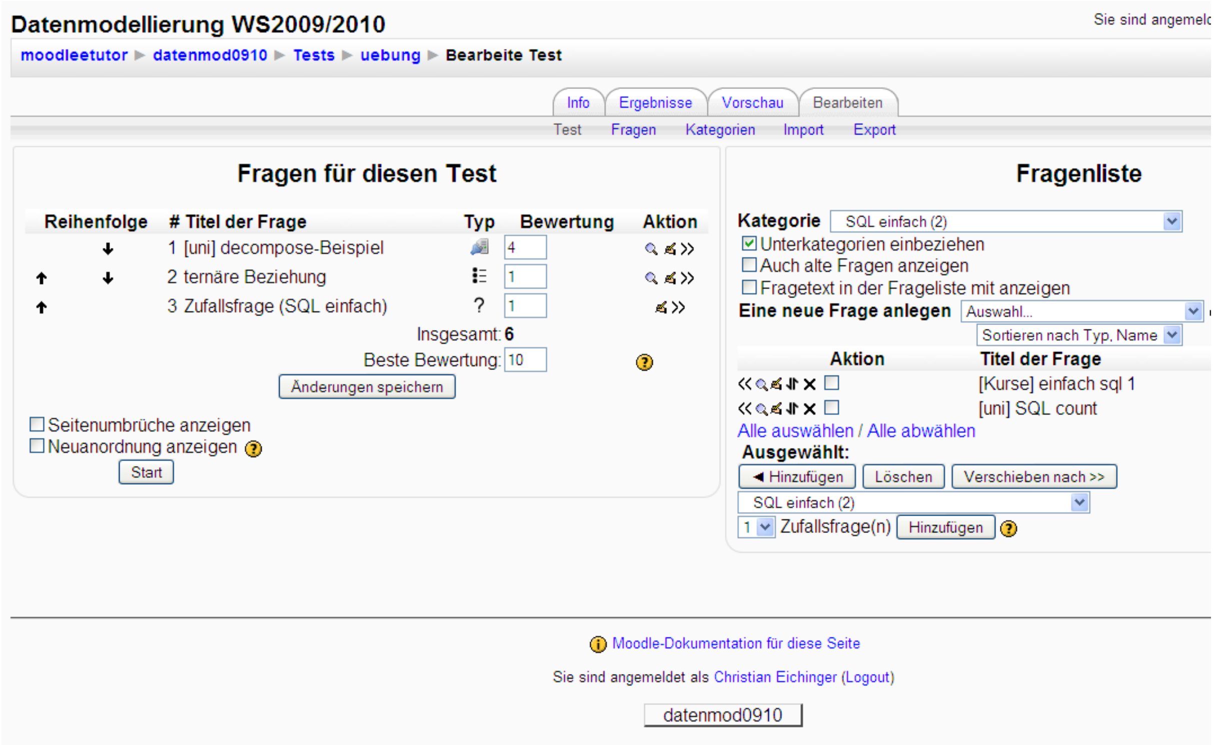


Abbildung 9 - Hinzufügen von Übungsaufgaben zum Test

3.2.2 Manuell geprüfte Übungsabgabe anlegen

Für Übungen mit Dateiupload wird zunächst eine neue Aktivität *Online – eine Datei hochladen* hinzugefügt. In der derzeitigen Implementierung des Moodle-Plugins zur Tutoren-Korrektur ist es nicht möglich, Übungsabgaben hoch zu laden, die aus mehreren Dateien bestehen. Dabei sollte ein Abgabetermin angegeben werden, ab dem die Abgabemaske gesperrt wird und Tutoren Bewertungen vornehmen können.

Anschließend können sich Teilnehmer einloggen und ihre Abgaben als pdf hochladen. Sobald die Deadline verstrichen ist, werden die Abgaben über das Cron-Skript von Moodle (manuell unter der URL *http://localhost/moodleetutor/admin/cron.php* ausführbar, die im Admin-Interface verlinkt ist) an die Tutoren verteilt. Wenn sich diese Einloggen können sie über den Block *DKE-Tutoren – Abgaben bewerten* für ihre zugewiesenen Abgaben Punkte vergeben und eine Rückmeldung hochladen. Diese kann anschließend von den Kursteilnehmern eingesehen werden.

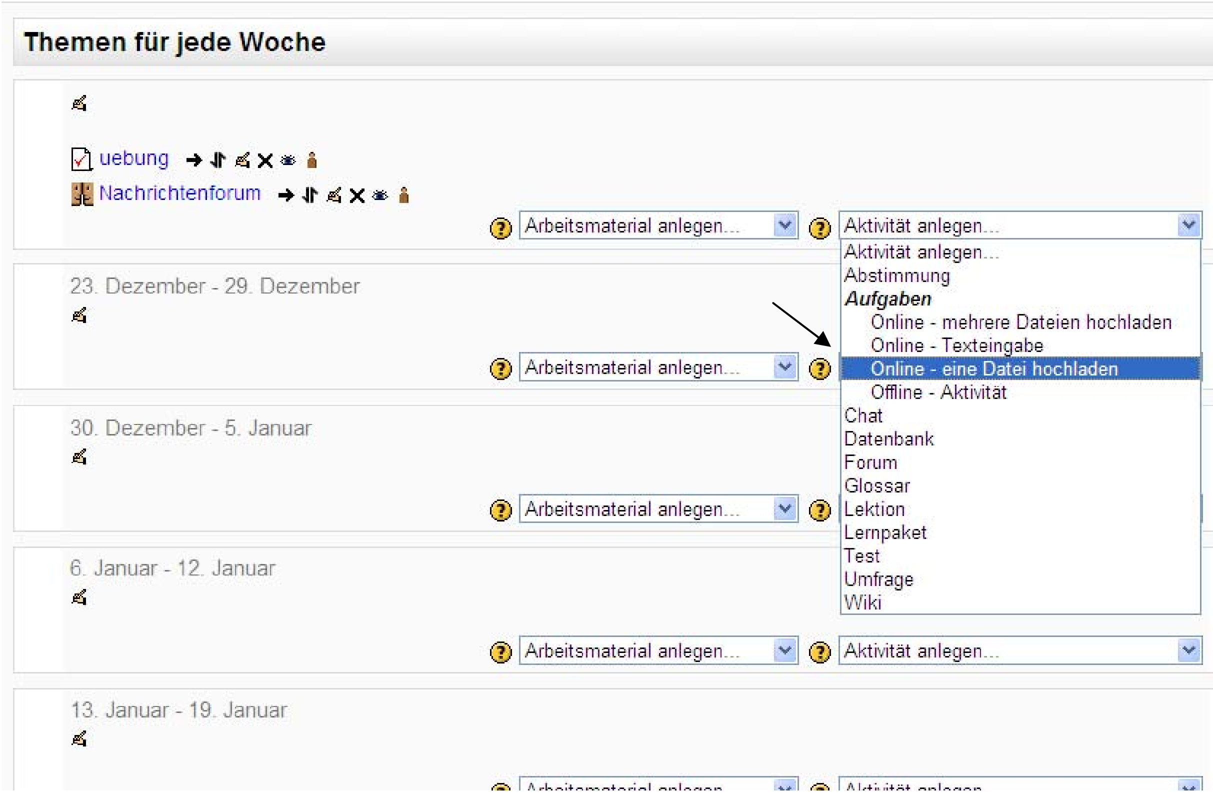


Abbildung 10 - Abgabeformular für manuell geprüfte Übungsabgabe anlegen

Im Erstellungsprozess kann eine Abgabefrist für die Übung angegeben werden.

Abbildung 11 - Abgabefrist für manuell korrigiertes Übungsbeispiel festlegen

4 Übungsbeispiel ausarbeiten (Teilnehmer)

Das Aussehen des Benutzerdialogs zum Ausarbeiten einer Übung variiert je nach Beispieltyp. In den folgenden Bildschirmausschnitten wird die Bearbeitung einer Decompose-Übung gezeigt.

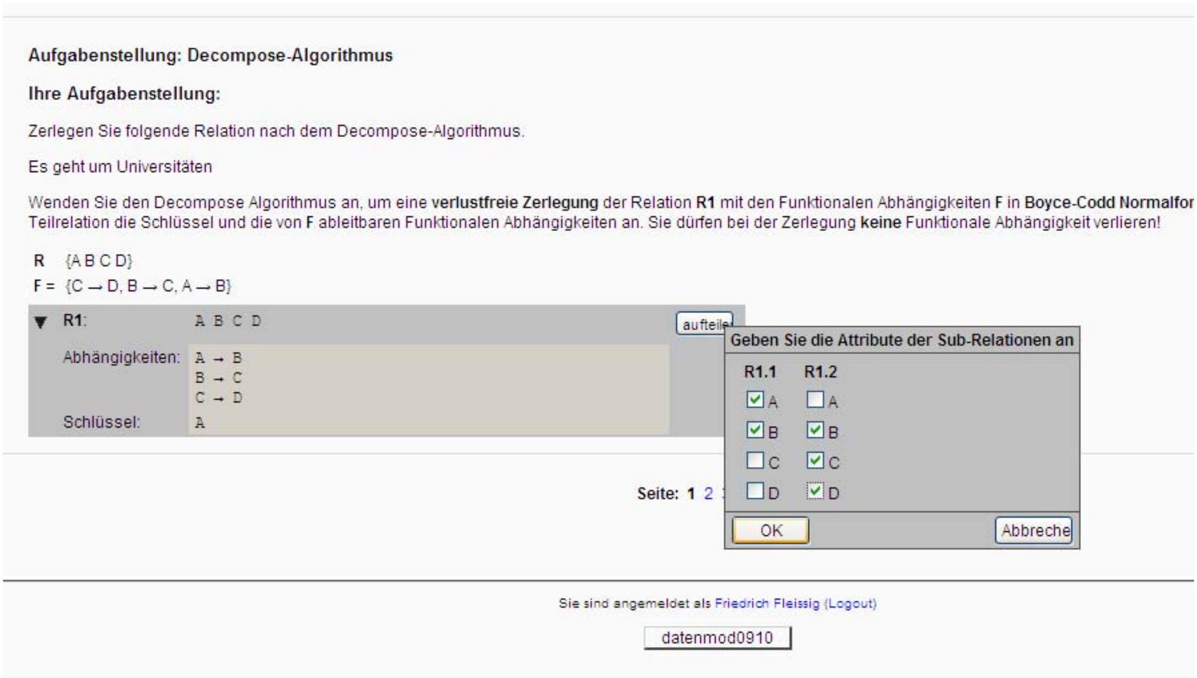


Abbildung 12 - Übungsbeispiel zum Decompose-Algorithmus ausarbeiten

Sobald alle Eingaben zur Erstellung einer Beispielausarbeitung vorgenommen wurden, kann über die Schaltfläche *Untersuchen* eine Rückmeldung vom eTutor-System angefordert werden. Bei der Funktion *Abgeben* wird zusätzlich zur automatischen Korrektur eine Punktzahl zur Übungsbewertung vergeben. Diese ist jedoch nicht endgültig, sondern kann innerhalb des Abgabezeitraums beliebig oft neu vergeben werden.

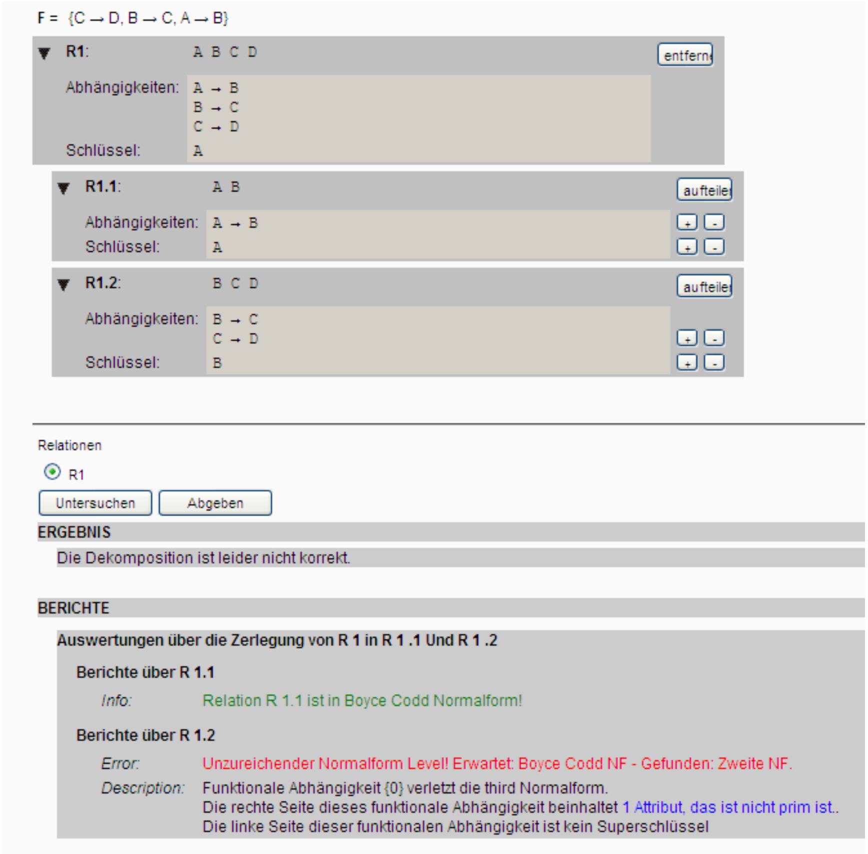


Abbildung 13 - Rückmeldung des eTutor-Expertenmoduls

Bei Übungsbeispielen, die manuell von Tutoren kontrolliert werden, können Kursteilnehmer ihre Übungsausarbeitung vorzugsweise im pdf-Format hochladen. Es ist möglich, die hochgeladene Datei innerhalb der Abgabefrist beliebig oft zu überschreiben.

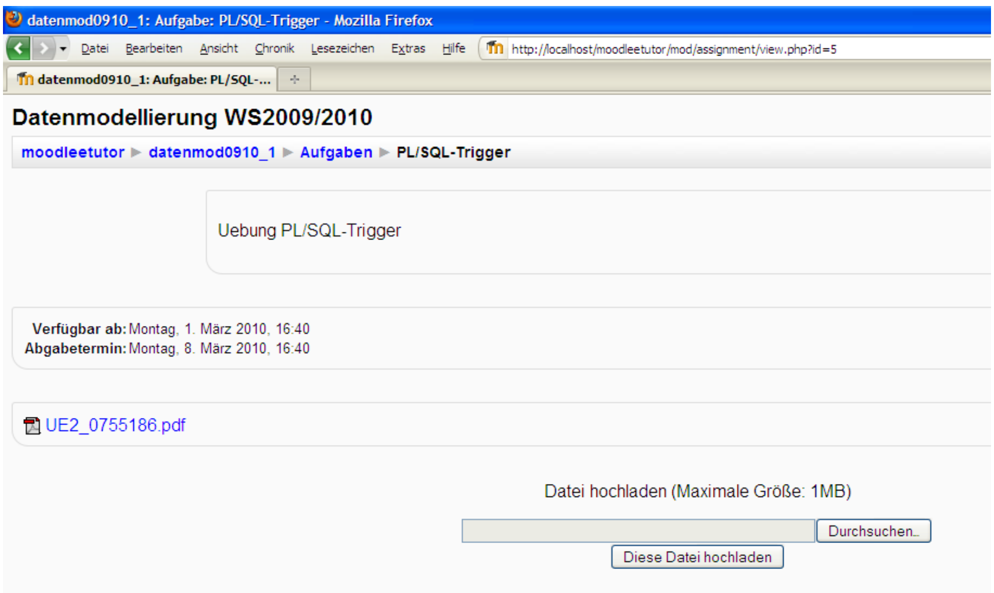
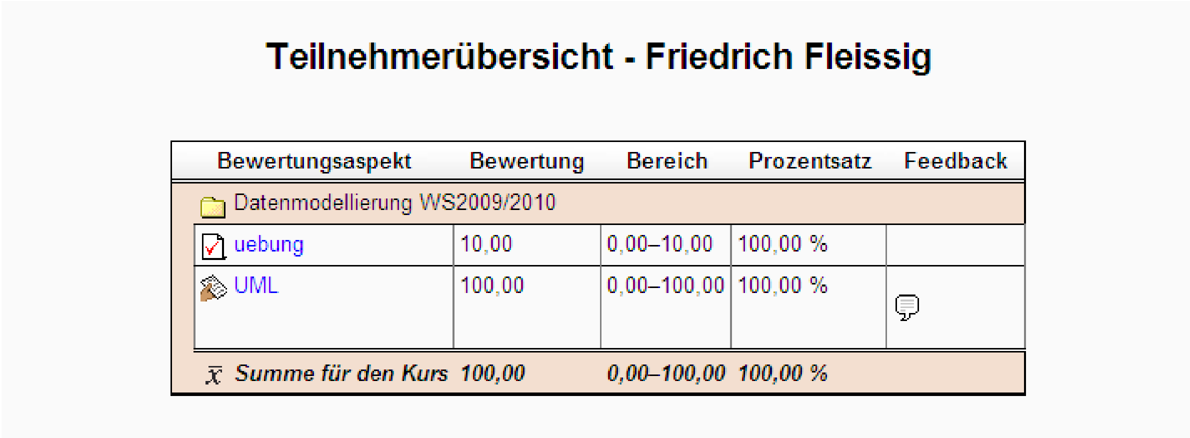


Abbildung 14 - Ausarbeitung für manuell korrigiertes Übungsbeispiel abgeben

Im Kursblock *Administration* können Kursteilnehmer eine Übersicht über ihre Übungs-Bewertungen ansehen. Bei Übungsbeispielen, die von Tutoren korrigiert wurden, verweist das Sprechblasen-Symbol der *Feedback*-Spalte auf die Rückmeldung des Tutors.



Teilnehmerübersicht - Friedrich Fleissig

Bewertungsaspekt	Bewertung	Bereich	Prozentsatz	Feedback
Datenmodellierung WS2009/2010				
uebung	10,00	0,00–10,00	100,00 %	
UML	100,00	0,00–100,00	100,00 %	
Summe für den Kurs	*100,00*	*0,00–100,00*	*100,00 %*	

Abbildung 15 - Übersicht zu Übungsbewertungen

5 Beispiel-Korrektur (Tutor)

Tutoren können eine Liste der ihnen zur Korrektur zugewiesenen Studentenabgaben über den Kursblock *DKE-Tutoren – Abgaben bewerten* abrufen. Wie in Abbildung 16 dargestellt, kann die Übungsausarbeitung des Studenten durch einen Klick auf den Aufgabennamen (Abgabe von frifle) geöffnet werden. Da es sich dabei um eine pdf-Datei handeln sollte, kann zur Rückmeldung an den Studenten vom Tutor kommentiert werden.

Abbildung 16 - zu bewertende Übungsabgaben anzeigen

Der über das *Bearbeiten*-Symbol in der Abgabenliste erreichbare Dialog dient dem Tutor zur Bewertung von Übungsabgaben. Neben einer erreichten Punktzahl, die beim Anlegen des Übungsbeispiels konfiguriert wurde (in Abbildung 17 zwischen 0 und 100 Punkte) und ganzzahlig sein muss, kann die kommentierte Übungs-Abgabedatei hochgeladen werden, die dem Studenten als Rückmeldung dient.

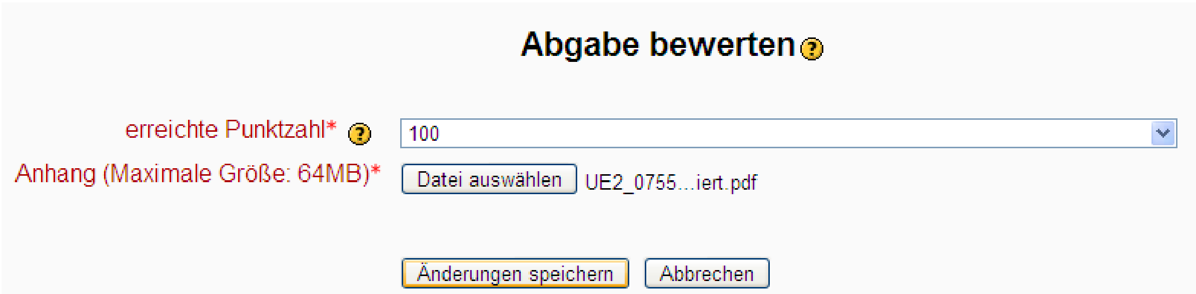


Abbildung 17 - Punktebewertung und Rückmeldung für Übungsabgaben

B. Installationshandbuch

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeausschnittverzeichnis

1 eTutor

Im ersten Teil der Installation wird die Installation des eTutor-Dispatchers und der Expertenmodule beschrieben.

1.1 Voraussetzungen

Für den Zugriff auf die Projektdateien ist ein Subversion-Client[17] notwendig. Zum Ausführen des Application-Servers wird JDK Version 6 benötigt.

1.2 Tomcat

Bei der Installation von Tomcat in Version 6 können die vorgegebenen Einstellungen belassen werden. Es muss jedoch darauf geachtet werden, dass Tomcat nicht als Windows-Dienst läuft, wenn der Servlet-Container aus Eclipse heraus gestartet werden soll. Andernfalls würden zwei Tomcat-Instanzen zur selben Zeit laufen und es würde zu einem Port-Konflikt kommen. Anschließend muss der Oracle-JDBC-Treiber (*ojdbc14.jar*) in den Ordner *lib* des Tomcat-Hauptverzeichnisses kopiert werden.

1.3 Eclipse

Als Entwicklungsumgebung sollte Eclipse in einer aktuellen Version ab 3.6 eingesetzt werden.

1.3.1 Startparameter setzen

Die Datei *eclipse.ini* im Eclipse-Installationsverzeichnis muss dahingehend geändert werden, dass Eclipse mittels Startparameter mehr verwendbarer Hauptspeicher zugewiesen wird. Durch die große Anzahl benötigter Plugins reicht der standardmäßig zugewiesene Speicher nicht aus:

```
--launcher.XXMaxPermSize 256m
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Xms256m
-Xmx1g
-XX:PermSize=128m
-XX:MaxPermSize=256m
-XX:-UseParallelGC
```

Codeausschnitt 1 - Startparameter in eclipse.ini[18]

1.3.2 Plugins installieren

Folgende Eclipse-Plugins werden zur Entwicklung des eTutor-Dispatchers und der Expertenmodule benötigt.

- Galileo - *http://download.eclipse.org/releases/galileo*
    - Web Tools Platform (WTP) für Webprojekte mit Tomcat-Integration
    - Subversion-Unterstützung

[17] https://subversion.dke.uni-linz.ac.at/etutor
[18] Pfad im Subversion-Repository:/implementation/Projekteinstellungen etutor/eclipse.ini

- Subversive Connector – *http://www.polarion.org/projects/subversive/download/eclipse/2.0/update-site/*
  - stellt Verbindung zum SVN-Repository her
- SpringIDE - *http://springide.org/updatesite*
  - unterstützt die Entwicklung mit Spring
- AJDT - *http://download.eclipse.org/tools/ajdt/35/update*
  - unterstützt die Programmierung mittels Aspekten
- Veloeclipse - *http://veloeclipse.googlecode.com/svn/trunk/update/*
  - Editor für Velocity-Templates

1.3.3 Java SDK integrieren

Damit der im folgenden Schritt eingestellte Application Server die richtige Java-Laufzeitumgebung nutzt, sollte ein aktuelles JDK unter Window/Preferences/Java/Installed JREs/Add manuell eingestellt werden. Hierbei ist es notwendig, das Verzeichnis der SDK-Installation anzugeben, zum Beispiel *C:\Programme\Java\jdk1.6.0_10*.

Anschließend sollten die Classpath-Einstellungen des Projekts dahingehend überprüft werden (rechte Maustaste auf ein Projekt/Java Build Path/Libraries), ob der Eintrag *JRE System Library* mit einem roten Kreuz versehen ist. In diesem Fall muss der Verweis auf die Java-Laufzeitumgebung angepasst werden, indem man den Eintrag löscht und über die Schaltfläche *Add Library* die JRE System Library *Workspace default JRE* auswählt und dadurch hinzufügt.

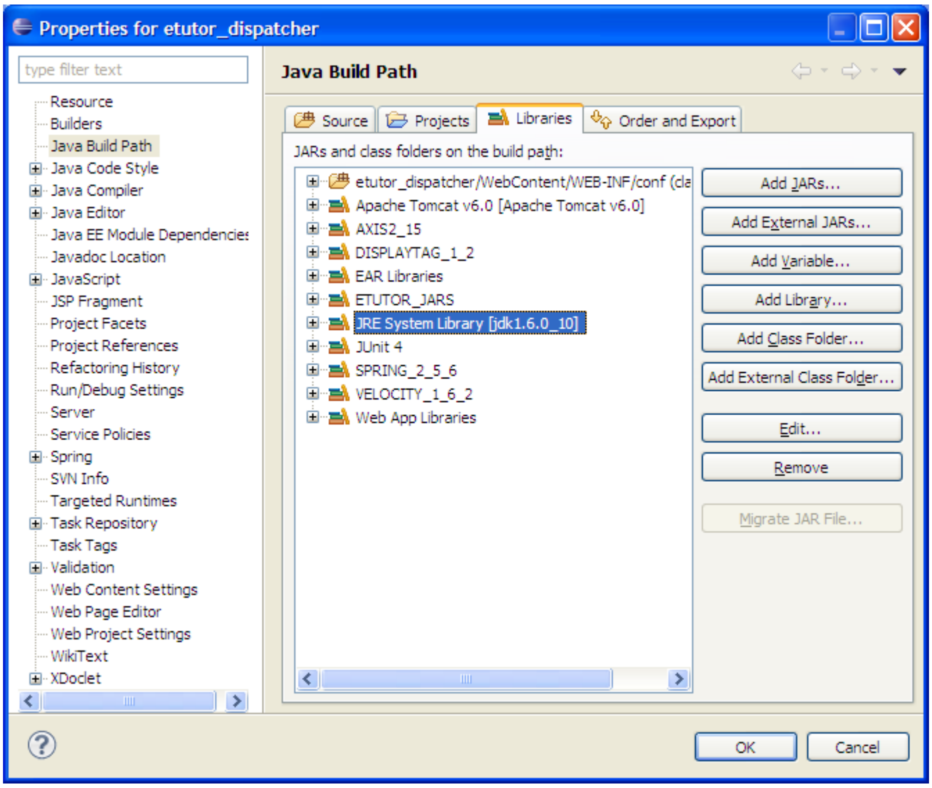


Abbildung 1 - korrekt eingebunde Java-Laufzeitumgebung

1.3.4 User-Libraries erstellen

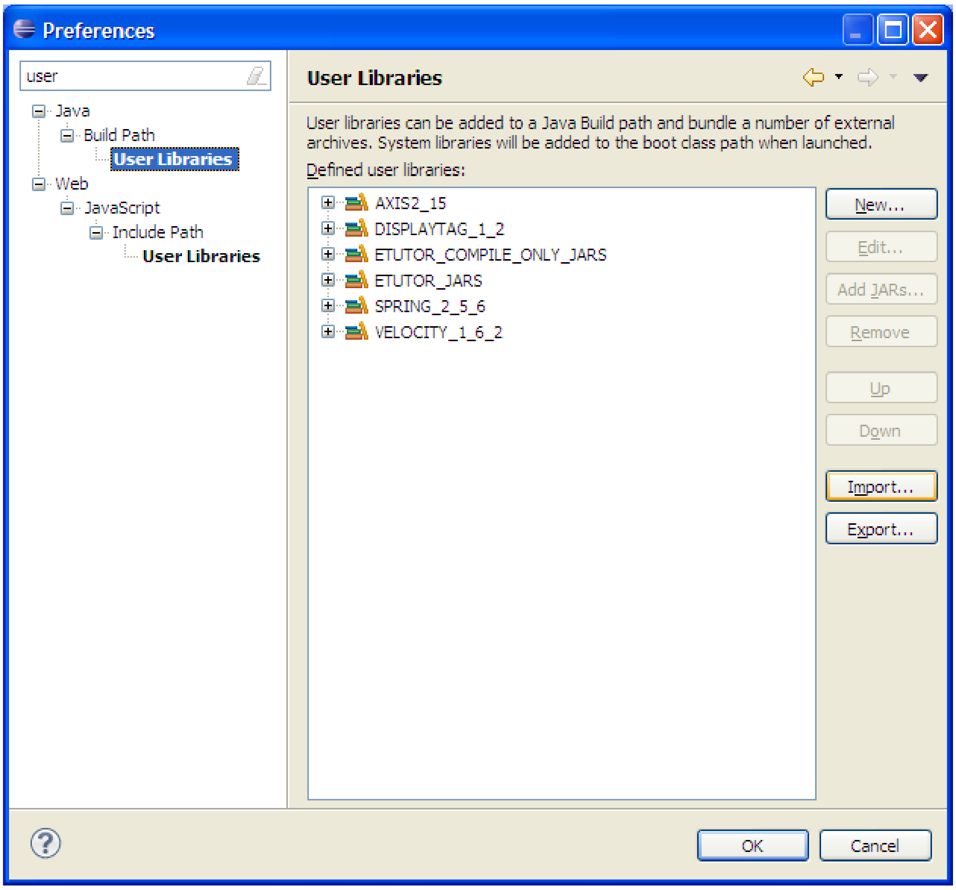
Um das eTutor-Java-Projekt compilieren und ausführen zu können, wird eine Reihe von Programmbibliotheken benötigt (unter anderem Axis, Spring und Velocity). Eclipse unterstützt das Zusammenfassen von jar-Dateien zu sogenannten User-Libraries, die im Projekt als Gesamtpaket eingebunden werden können.

Um die für das eTutor-Projekt benötigten User-Libraries zu importieren, müssen zunächst in der Datei *lib.userlibraries* alle absoluten Pfade zum Ordner, in dem die libraries liegen (Repository), angepasst werden:

```
…
<archive
      path="D:/EigeneDateien/Uni/DA/repository-da/axis2-1.5-
            bin/lib/activation-1.1.jar"
/>
…
```

Codeausschnitt 2 - Repository-Pfad bei lib.userlibraries[19] anpassen

Die eingetragenen Libraries können im nächsten Schritt unter *Window/Preferences – Java/Build Path/User Libraries/Import* durch Auswahl der editierten Datei hinzugefügt werden.



Codeausschnitt 3 - User-Libraries importieren

[19] /implementation/Projekteinstellungen etutor/lib.userlibraries

1.3.5 Application-Server hinzufügen

Die Tomcat-Installation kann unter den Eclipse-Einstellungen (*Window/Preferences*) unter *Server/Runtime Environment/Add* hinzugefügt werden. Aus der Liste der Server muss Apache Tomcat v6.0 gewählt werden. Im nächsten Schritt wird das Installationsverzeichnis abgefragt. Außerdem muss darauf geachtet werden, unter JRE die manuell hinzugefügte JDK-Installation auszuwählen.

Den Server kann man anschließend als Projekt einbinden, indem man in der View *Servers* (erreichbar unter dem Menüeintrag *Window/Show View/Servers*) mit der rechten Maustaste in den weißen Bereich klickt. Im dann erscheinenden Kontextmenü wählt man den Eintrag *New/Server* und in der daraufhin erscheinenden Liste möglicher Servertypen den Eintrag *Tomcat v6.0 Server*.

1.3.6 Projekte importieren

Folgende Projekte aus dem svn-Repository (*implementation*-Verzeichnis) werden benötigt:

- etutor_dispatcher
- etutor_dlg
- etutor_fd
- etutor_jdbc
- etutor_ra
- etutor_sql
- etutor_xq

Diese können innerhalb der *SVN Repository Exploring-Perspective* ausgecheckt werden.

Anschließend werden Fehlermeldungen angezeigt, wonach sich Projekte wechselseitig importieren. Der eTutor-Dispatcher muss die Expertenmodule einbinden, da ihre Class-Dateien mitdeployed und in der Spring-Config referenziert werden. Expertenmodule benötigen wiederum mehrere Klassen aus dem Dispatcher. Aus diesem Grund werden vom Compiler Buildfehler ausgegeben:

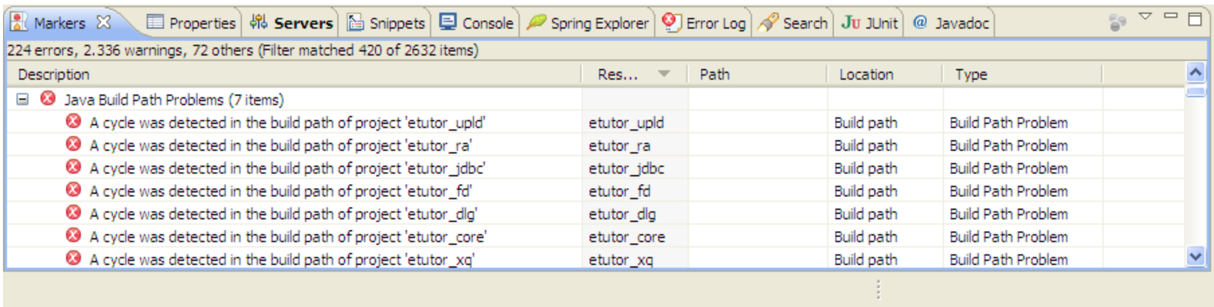


Abbildung 2 - Buildfehler wechselseitig eingebundener Projekte

Um diese Fehler auf Warnings umzustellen, muss in den Eclipse-Einstellungen unter *Java/Compiler/Building* der Wert *Circular dependencies* auf *Warning* gesetzt werden.

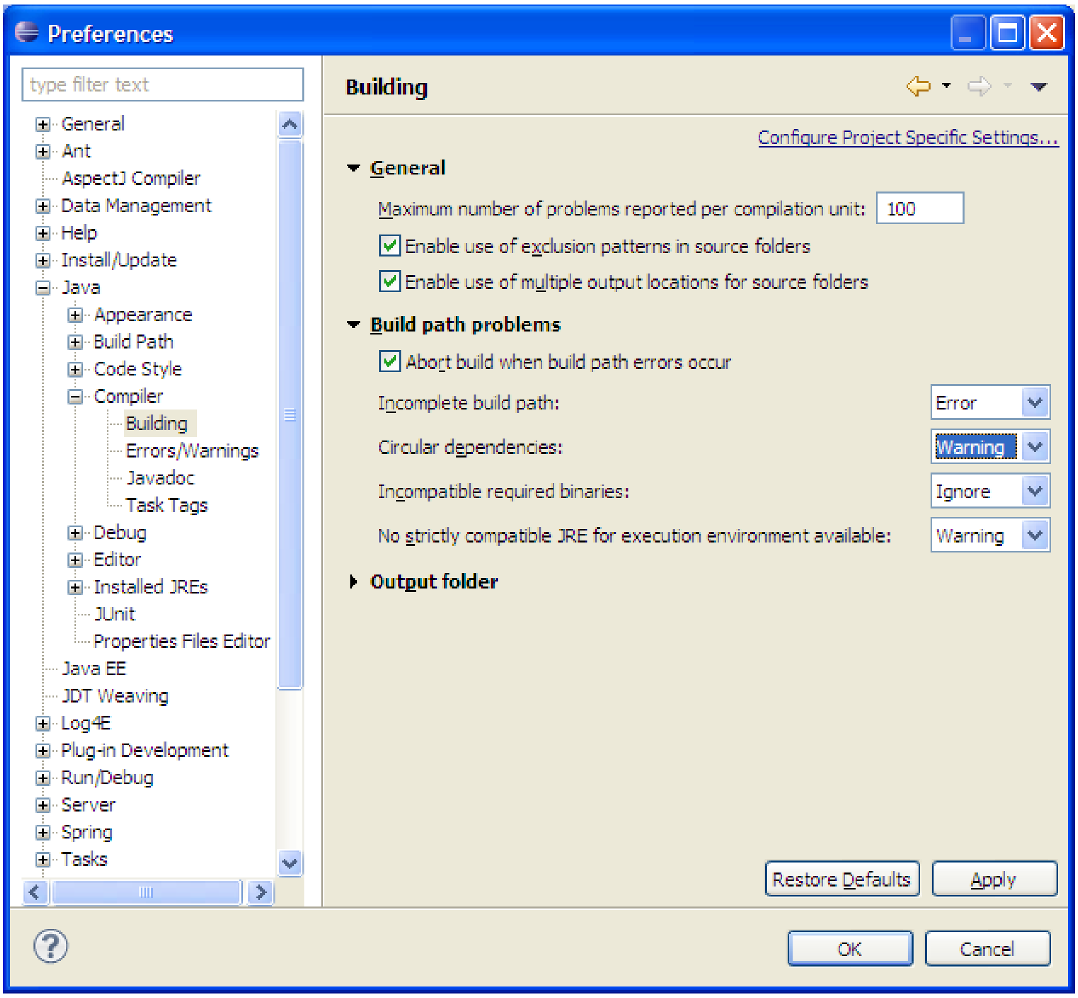


Abbildung 3 - Buildfehler auf Warning umstellen

Im nächsten Schritt müssen die im Quellcode angegebenen Datenbankverbindungen angepasst werden. Diese befinden sich in den Dateien:

- etutor_dispatcher - *etutor.resources.modules.jdbc: jdbc.properties*
- etutor_dlg - *ooc-jndi.xml*
- etutor_fd - *etutor.modules.rdbd.RDBDConstants.java*
- etutor_jdbc - *resources/modules/jdbc/jdbc.properties*
- etutor_ra - *etutor.modules.ra2sql.RAConstants.java*
- etutor_sql - *etutor.modules.sql.SQLConstants.java*
- etutor_xq - *ooc-jndi.xml*

In der Datei etutor_dispatcher - *etutor.resources.modules.jdbc: jdbc.properties* müssen zudem die Pfade zu den Logdateien angepasst werden:

```
umfh.pattern = D:/EigeneDateien/Uni/DA/tomcat-da/webapps/etutor/log/UserMessages%u.log
…
```

```
cmfh.pattern = D:/EigeneDateien/Uni/DA/tomcat-da/webapps/etutor/log/CoreMessages%u.log
```

Codeausschnitt 4 - jdbc.properties[20]

Das *etutor-dispatcher*-Webprojekt kann nun auf den Tomcat-Server kopiert (deployed) werden.

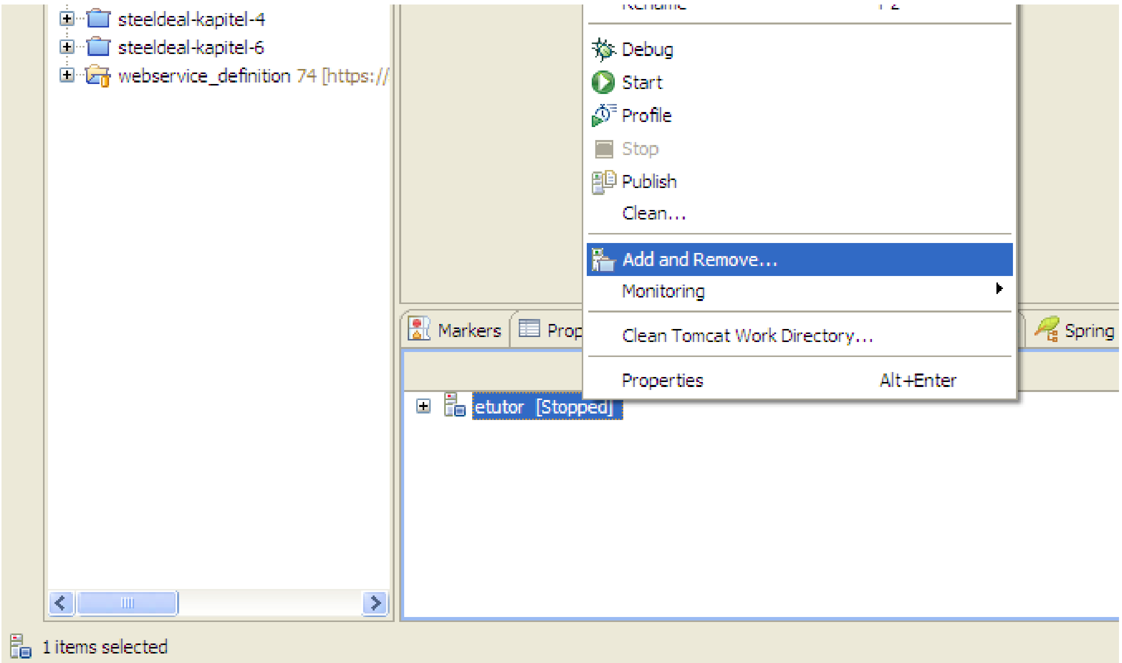


Abbildung 4 - Projekt bei Server hinzufügen

Dazu wird wie in Abbildung 4 dargestellt der Add and Remove-Dialog aufgerufen. In diesem wird das *etutor_dispatcher*-Projekt von der available-Ansicht auf die configured-Ansicht verschoben.

Außerdem müssen in den Tomcat-Einstellungen Datenbankverbindungsparameter angegeben werden. Dazu wird in der Eclipse-Ansicht im Projekt *Servers* die Datei *server.xml* editiert. Die Konfiguration für das Projekt *etutor_dispatcher* muss die angegebenen Inhalte umfassen. Die Einstellungen zu den angegebenen Datenbanken müssen jedoch auf die jeweilige Systemumgebung angepasst werden.

```
<Context
      docBase="etutor_dispatcher"
      path="/etutor"
      reloadable="true"
      source="org.eclipse.jst.jee.server:etutor_dispatcher">
      <Resource
            auth="Container"
            mail.from="etutor@dke.uni-linz.ac.at"
            mail.smtp.auth="true"
            mail.smtp.host=""
            mail.smtp.user=""
            name="mail/etutor/coreMS"
            password=""
            type="javax.mail.Session"
      />
```

[20] /implementation/etutor_dispatcher/src/etutor/resources/modules/jdbc/jdbc.properties

```
<Resource
      auth="Container"
      driverClassName="oracle.jdbc.driver.OracleDriver"
      logAbandoned="true"
      maxActive="30"
      maxIdle="30"
      maxWait="15000"
      name="jdbc/etutor/coreDS"
      password=""
      removeAbandoned="true"
      removeAbandonedTimeout="60"
      type="javax.sql.DataSource"
      url="jdbc:oracle:thin:@localhost:1500:etutor"
      username=""
/>
<Resource
      auth="Container"
      driverClassName="oracle.jdbc.driver.OracleDriver"
      logAbandoned="true"
      maxActive="10"
      maxIdle="10"
      maxWait="15000"
      name="jdbc/etutor/modules/rdbd/rdbdDS"
      password=""
      removeAbandoned="true"
      removeAbandonedTimeout="60"
      type="javax.sql.DataSource"
      url="jdbc:oracle:thin:@localhost:1500:etutor"
      username=""
/>
<!-- xq-modul -->
<Resource
      auth="Container"
      driverClassName="oracle.jdbc.driver.OracleDriver"
      logAbandoned="true"
      maxActive="10"
      maxIdle="10"
      maxWait="15000"
      name="jdbc/etutor/modules/xquery/xqDS"
      password=""
      removeAbandoned="true"
      removeAbandonedTimeout="60"
      type="javax.sql.DataSource"
      url="jdbc:oracle:thin:@localhost:1500:etutor"
      username=""
/>
<!-- datalog-modul -->
<Resource
      auth="Container"
      driverClassName="oracle.jdbc.driver.OracleDriver"
      logAbandoned="true"
      maxActive="10"
      maxIdle="10"
      maxWait="15000"
      name="jdbc/etutor/modules/datalog/datalogDS"
      password=""
      removeAbandoned="true"
      removeAbandonedTimeout="60"
      type="javax.sql.DataSource"
      url="jdbc:oracle:thin:@localhost:1500:etutor"
      username=""
```

```
      />
</Context>
```

Codeausschnitt 5 - server.xml[21] anpassen

An dieser Stelle werden einige serverspezifische Einstellungen vorgenommen.

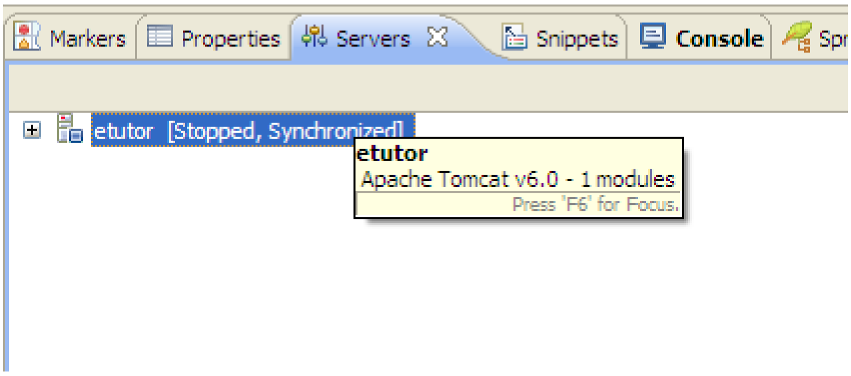


Abbildung 5 - Servereinstellungen öffnen

Ein Doppelklick auf den Server in der Serverliste öffnet den Einstellungsdialog. Die Aufrufparameter können im Dialog im Einstellungsfeld *Open Launch Configuration* editiert werden.

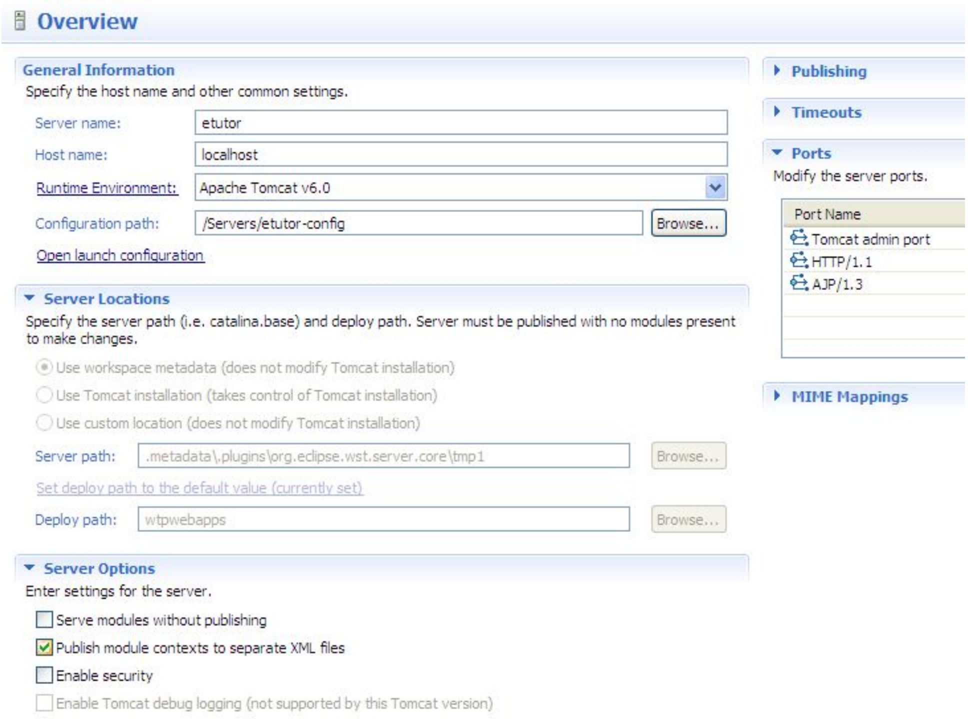


Abbildung 6 - Servereinstellungen

Wenn Tomcat mit den eingestellten Projekten auch extern, also außerhalb der Eclipse-Instanz, lauffähig sein soll, muss unter der Einstellungskategorie *Server Options* die Option *Publish module contexts to seperate XML files* aktiviert werden.

---

[21] /implementation/Servers/etutor-config/server.xml

Um stabil zu laufen, sollte Tomcat mehr Hauptspeicher zugewiesen werden, als ihm standardmäßig zur Verfügung steht. Der zugewiesene Speicher wird über die Aufrufparameter von Tomcat angepasst.

Im Textfeld VM arguments wird folgender Übergabeparameter unten angefügt:

```
-Xmx512m
```

Codeausschnitt 6 - Tomcat Startparameter

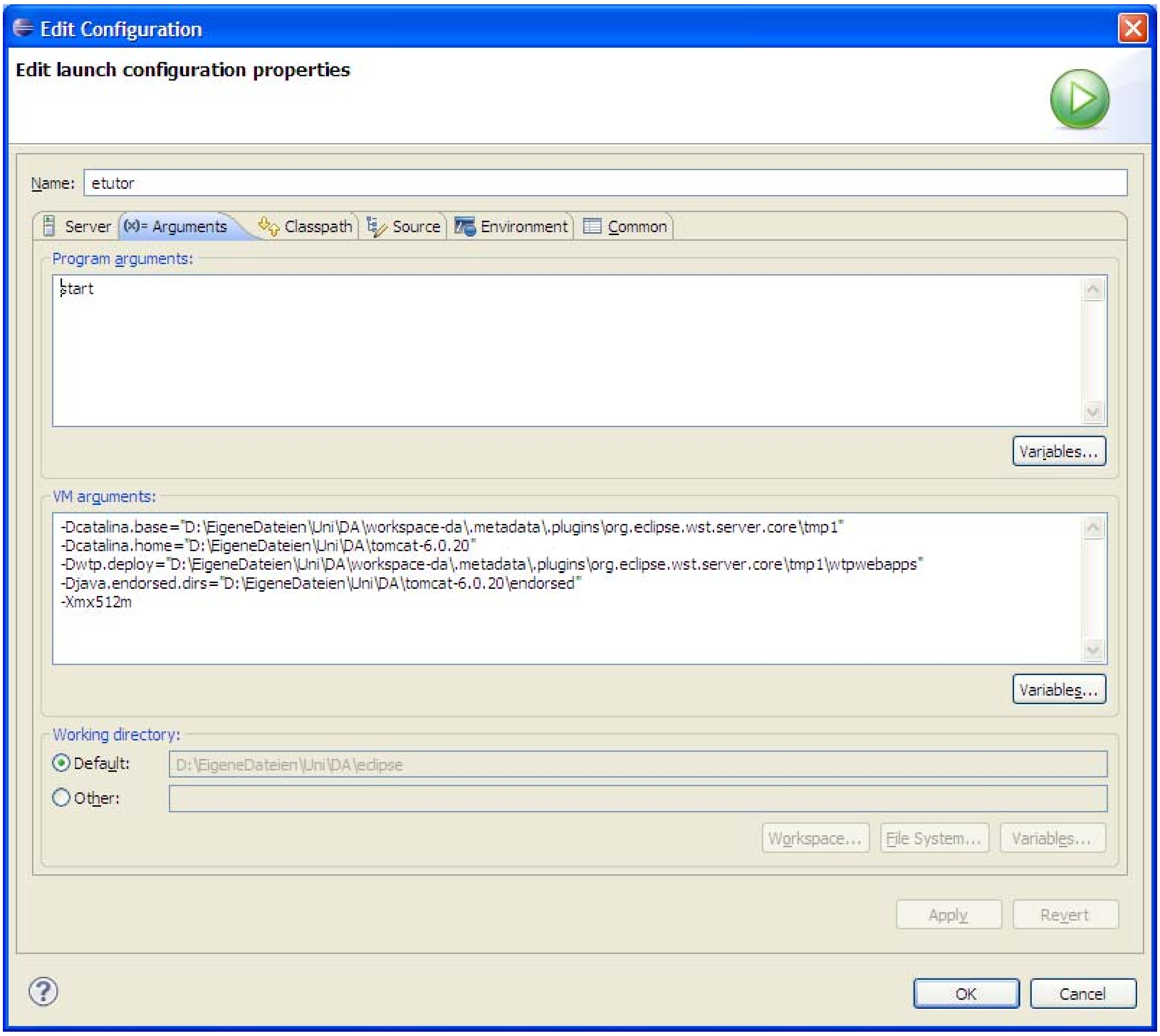
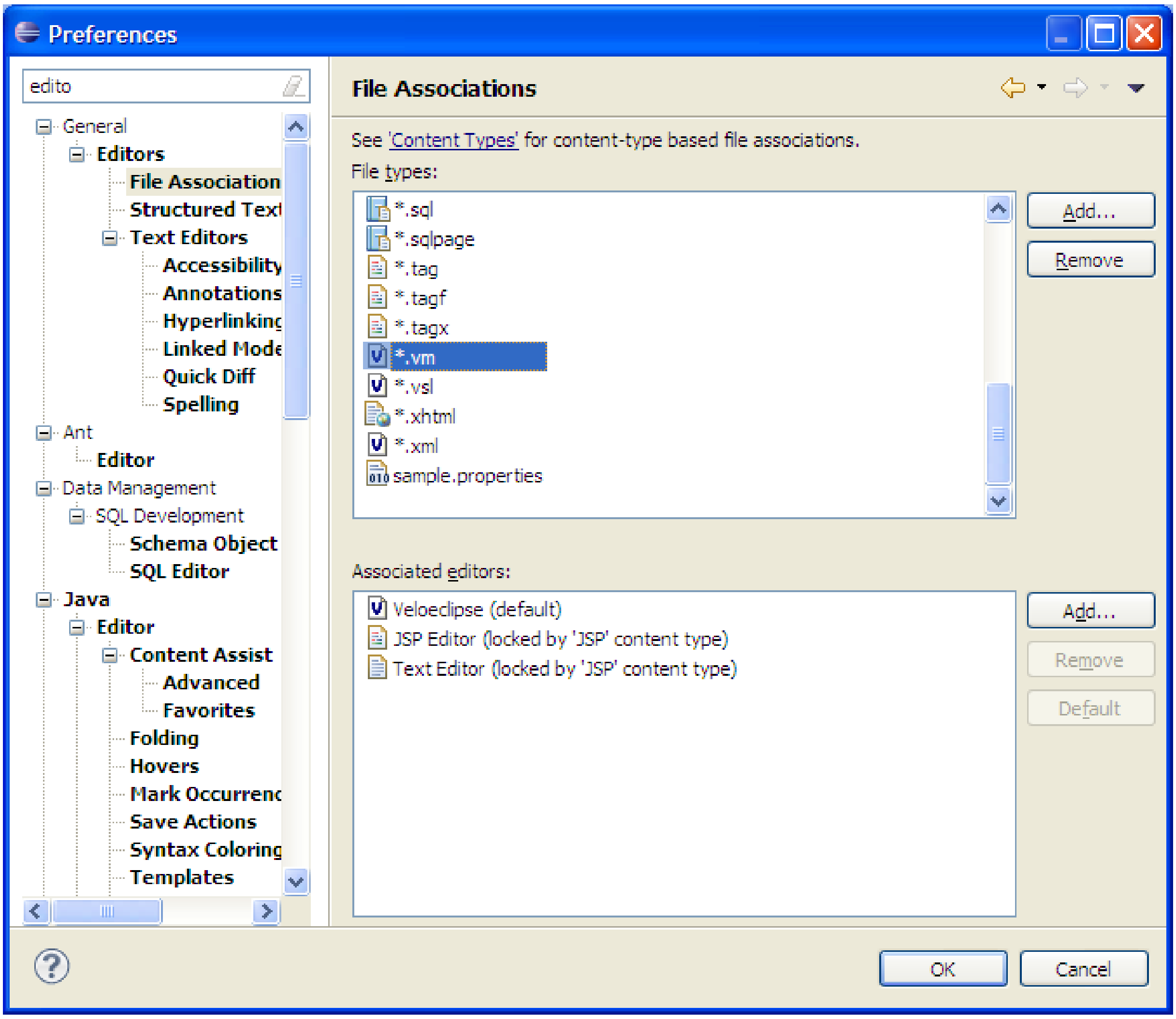


Abbildung 7 - Startparameter bei Tomcat einstellen

1.3.7 Dateizuordnungen für Velocity anpassen

Das Plugin Veloeclipse stellt einen für Velocity-Templates angepassten Editor bereit. Damit dieser beim Öffnen einer Velocity-Datei gestartet wird, muss die Zuordnung des Dateityps angepasst werden. Unter *Window/Preferences - General/Editors/File Association* kann diese Einstellung für die Dateieendung .vm vorgenommen werden, indem in der Liste *Associated editors* der Eintrag *Veloeclipse* als Standardeintrag (*default*) gesetzt wird.



Codeausschnitt 7 - Dateityp vm soll mit dem Plugin Veloeclipse geöffnet werden

1.3.8 Unit-Tests ausführen

Um einige Funktionalitäten der konfigurierten Projekte zu testen, sollten die Tests des Projekts *etutor_dispatcher* ausgeführt werden. Über die Testklasse *test.etutor.core.ws.TestSuite* werden alle Unit-Tests gestartet.

2 Moodle

Für den Betrieb von Moodle werden der Apache Webserver mit PHP-Modul in Kombination mit einer MySQL-Datenbank eingesetzt.

2.1 XAMPP

Die Installation von XAMPP wird für Moodle in der Moodle-Anleitung (*http://docs.moodle.org/en/Windows_installation_using_XAMPP*) beschrieben. Dabei sollte XAMPP in Version 1.7.1[22] eingesetzt werden, da bei einer Moodle-Installation mit der neueren XAMPP-Version 1.7.3 nicht näher untersuchte PHP-Fehlermeldungen aufgetreten sind.

Im Besonderen sollte darauf geachtet werden, dass in der PHP-Konfigurationsdatei *php.ini* im Ordner *XAMPP-Installationsverzeichnis/php* folgende Zeilen aktiviert sind, also kein Strichpunkt am Zeilenbeginn steht:

```
extension=php_curl.dll
…
extension=php_openssl.dll
```

Codeausschnitt 8 - php.ini: Module curl und openssl aktivieren

Die Standardkonfiguration der MySQL-Datenbank sollte dahingehend geändert werden, dass ein neuer Datenbankbenutzer mit Passwort eingerichtet wird. Dieser wird für die Installation der Moodle-Datenbank verwendet.

2.2 Eclipse

Für die Entwicklung von Moodle-Erweiterungen eignet sich Eclipse in Kombination mit dem Web Tools Platform-Plugin. Falls Moodle nur installiert werden soll, kann man diesen Abschnitt überspringen und mit Kapitel 2.3 fortsetzen.

2.2.1 Startparameter setzen

Das Einstellen der korrekten Startparameter wird in Kapitel 1.3.1 - Startparameter setzen beschrieben.

2.2.2 Plugins installieren

Folgende Plugins werden zur Entwicklung des etutor-dispatchers und der Expertenmodule benötigt:

- Galileo - *http://download.eclipse.org/releases/galileo*
    - Web Tools Platform (WTP) für PHP-Unterstützung
    - Subversion-Unterstützung
- Subversive Connector - *http://www.polarion.org/projects/subversive/download/eclipse/2.0/update-site/*

[22] http://sourceforge.net/projects/xampp/files/XAMPP%20Windows/1.7.1/xampp-win32-1.7.1-installer.exe/download

- stellt die Verbindung zum SVN-Repository her

2.2.3 PHP-Debugger einrichten

Um das Verhalten des Moodle-Codes zu untersuchen, bietet sich die Einrichtung eines PHP-Compilers an. Dazu müssen einige Einstellungen in der PHP-Konfigurationsdatei php.ini (im XAMPP-Verzeichnis unter php/php.ini) vorgenommen werden. Der in der XAMPP-Installation standardmäßig aktive Zend-Debugger wird durch die XDebug-Erweiterung ersetzt.

Im ersten Schritt werden daher der Zend-Debugger und die php_xdebug-Erweiterung durch Voranstellen eines Strichpunkts auskommentiert.

```
;extension=php_xdebug.dll
…

[Zend]
;zend_extension_ts =
"D:\EigeneDateien\Uni\DA\xampp\php\zendOptimizer\lib\ZendExtensionManager.d
ll"
;zend_extension_manager.optimizer_ts =
"D:\EigeneDateien\Uni\DA\xampp\php\zendOptimizer\lib\Optimizer"
;zend_optimizer.enable_loader = 0
;zend_optimizer.optimization_level=15
;zend_optimizer.license_path =
```

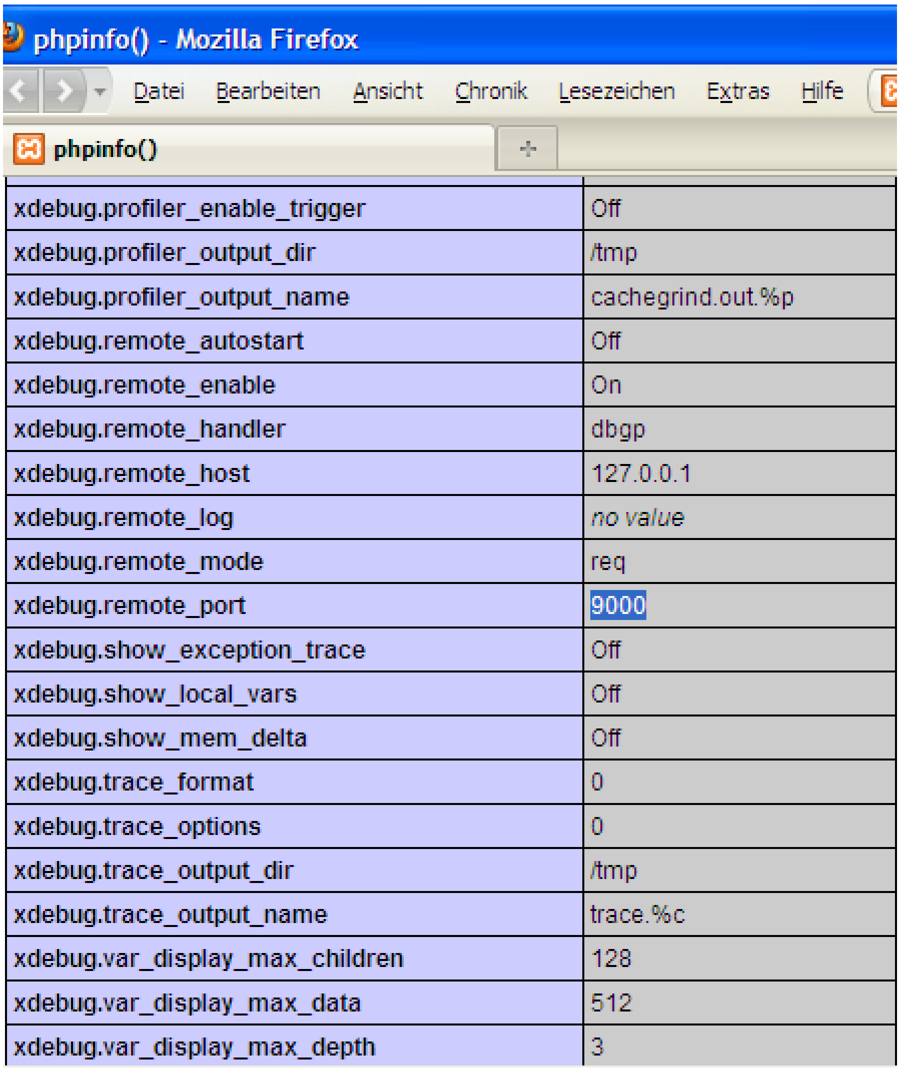
Codeausschnitt 9 - php.ini: Zend-Debugger deaktivieren

Um das XDebug-Modul zu aktivieren, muss wiederum die Datei php.ini editiert werden; Dazu werden die Kommentarzeichen am Zeilenbeginn der vorbereiteten Einstellungsdatei entfernt. Die Einstellung *xdebug.remote_host* gibt dabei die IP-Adresse des Moodle-Webservers an, während für den Wert *xdebug.remote_port* ein beliebiger freier Port festgelegt werden kann, über den mit dem Debugger kommuniziert werden kann.

```
[XDebug]
;; Only Zend OR (!) XDebug
zend_extension_ts="D:\EigeneDateien\Uni\DA\xampp\php\ext\php_xdebug.dll"
xdebug.remote_enable=true
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
xdebug.remote_handler=dbgp
;xdebug.profiler_enable=1
;xdebug.profiler_output_dir="D:\EigeneDateien\Uni\DA\xampp\tmp"
```

Codeausschnitt 10 - php.ini: XDebug-Erweiterung konfigurieren

Um zu prüfen, ob die Einstellungen korrekt übernommen wurden, kann man die PHP-Funktion phpinfo() verwenden. Diese ist bei der XAMPP-Installation bereits eingerichtet und bei einem lokalen Webserver auf Port 80 unter *http://localhost/xampp/phpinfo.php* erreichbar.



xdebug.profiler_enable_trigger	Off
xdebug.profiler_output_dir	/tmp
xdebug.profiler_output_name	cachegrind.out.%p
xdebug.remote_autostart	Off
xdebug.remote_enable	On
xdebug.remote_handler	dbgp
xdebug.remote_host	127.0.0.1
xdebug.remote_log	*no value*
xdebug.remote_mode	req
xdebug.remote_port	9000
xdebug.show_exception_trace	Off
xdebug.show_local_vars	Off
xdebug.show_mem_delta	Off
xdebug.trace_format	0
xdebug.trace_options	0
xdebug.trace_output_dir	/tmp
xdebug.trace_output_name	trace.%c
xdebug.var_display_max_children	128
xdebug.var_display_max_data	512
xdebug.var_display_max_depth	3

Abbildung 8 - Überprüfen der Debug-Einstellungen mithilfe des Befehls phpinfo()

Im nächsten Schritt müssen die Eclipse-Debugger-Einstellungen mit den im letzten Schritt vorgenommenen XAMPP-Einstellungen abgestimmt werden. Unter *Window/Preferences – PHP/Debug/Installed Debuggers* werden die von der Eclipse-Installation unterstützten Debugger aufgelistet. Nach der Auswahl des Eintrags *XDebug* kann der Einstellungsdialog über den Button *Configure* geöffnet werden.

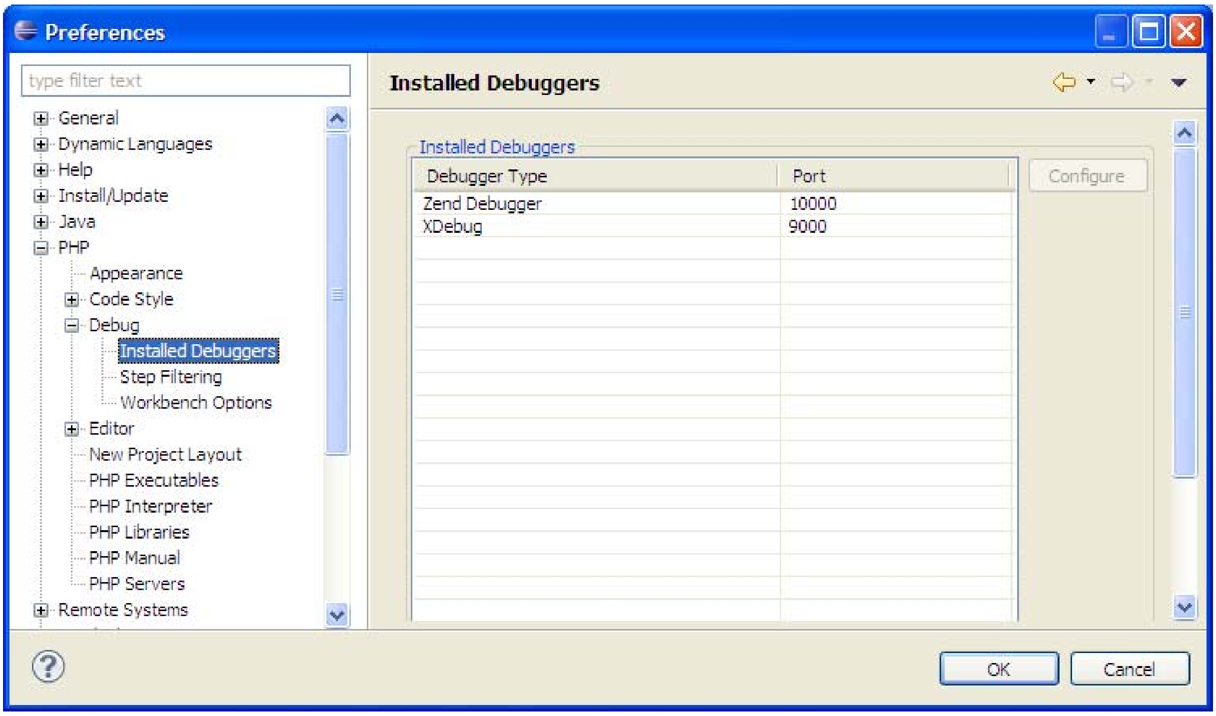


Abbildung 9 - PHP-Debugger: Port einstellen

Der gewählte Port muss dabei mit der Option *xdebug.remote_port* der Datei php.ini übereinstimmen.

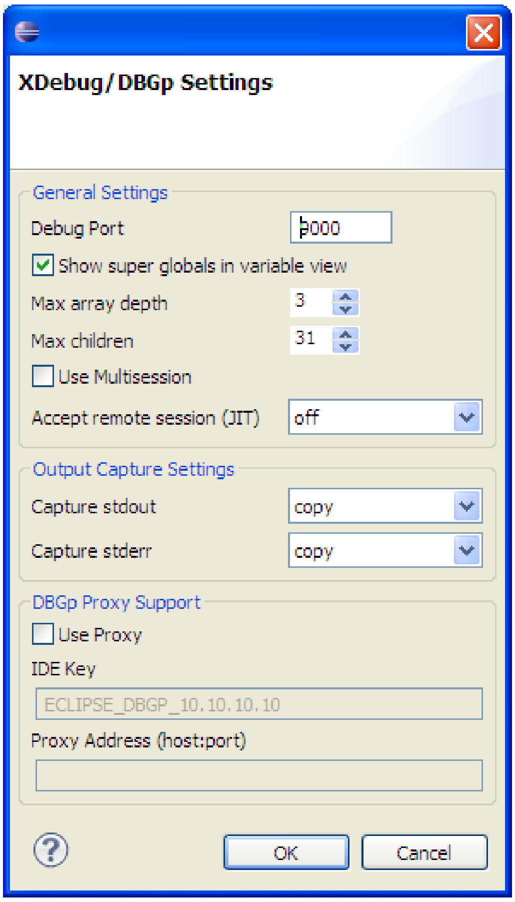


Abbildung 10 - PHP-Debugger: Port konfigurieren

Der Moodle-Webserver kann in den Eclipse-Einstellungen unter *PHP/PHP Servers* gesetzt werden.

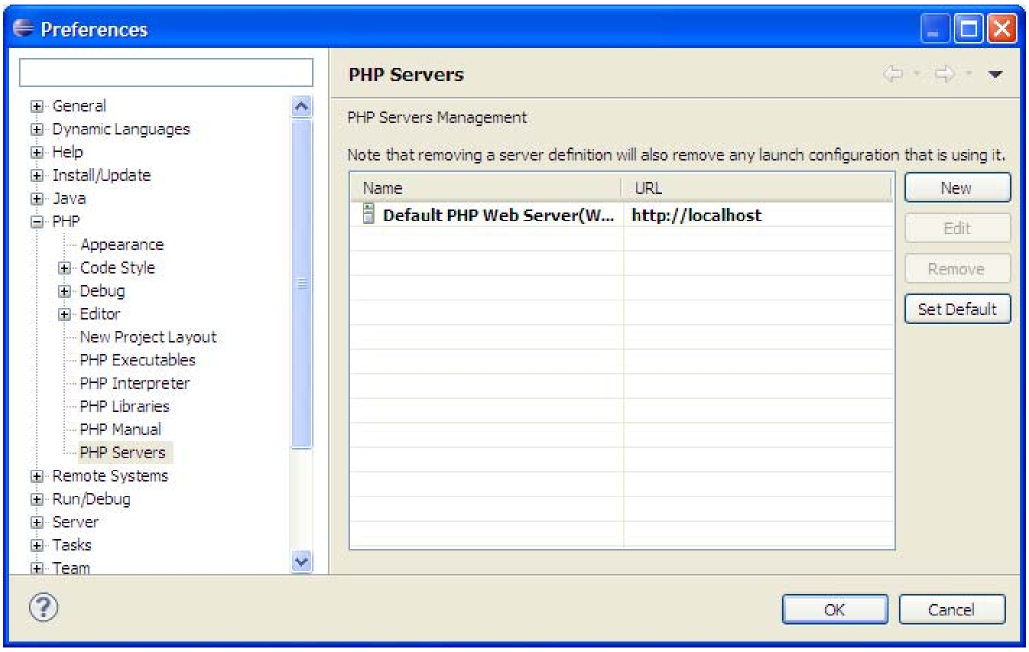


Abbildung 11 - PHP-Debugger: Serververbindung einrichten

Zum Starten des Debugvorgangs dient der unter *Run/Debug Configurations…* erreichbare Eclipse-Dialog. Die Einstellungen sollten dabei wie auf den Screenshots dargestellt vorgenommen werden. Als Debugger wird *XDebug* und als Webserver der im letzten Schritt konfigurierte *Default PHP Webserver* ausgewählt. Die Einstellung *File* gibt an, welche Seite beim Starten des Debug-Vorgangs ausgeführt werden soll.

Das PHP-Skript /moodleetutor/question/type/etutor/etutor_admin/engine_list.php listet Informationen zur eTutor-Verbindung auf.

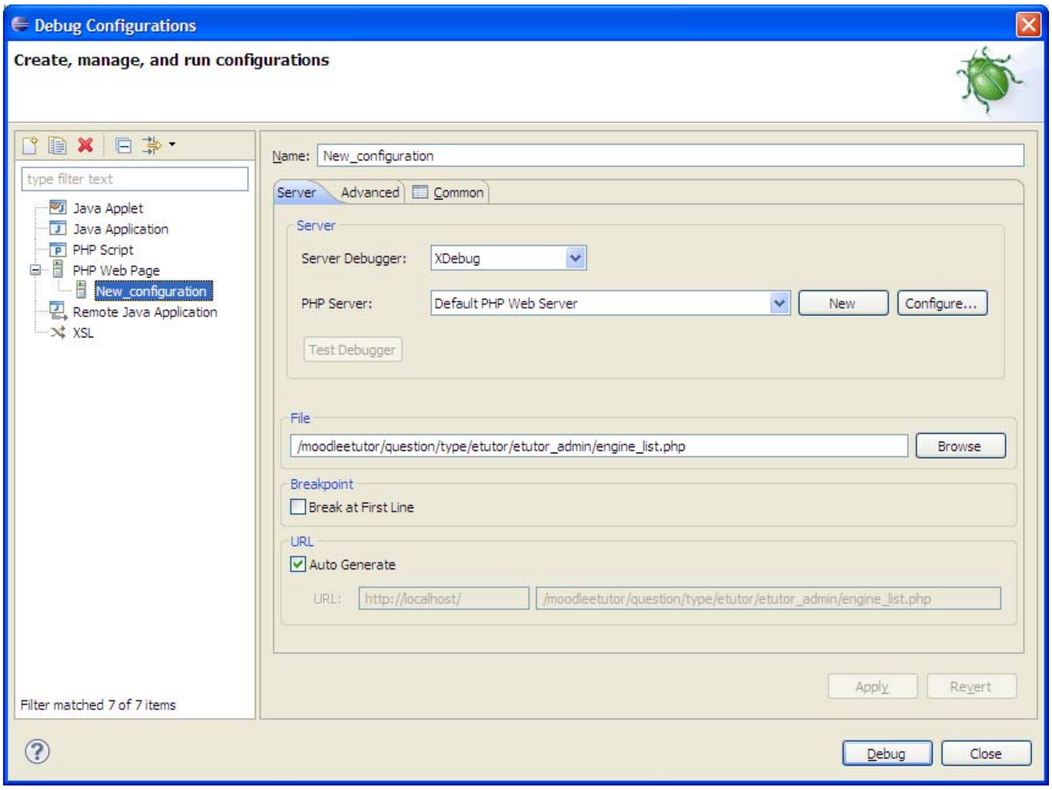


Abbildung 12 - Debugging-Einstellungen 1

Im Karteireiter *Advanced* sollte die Einstellung *Debug All Pages* gesetzt sein, damit auf Breakpoints aller PHP-Skripten im Moodle-Verzeichnis reagiert wird.

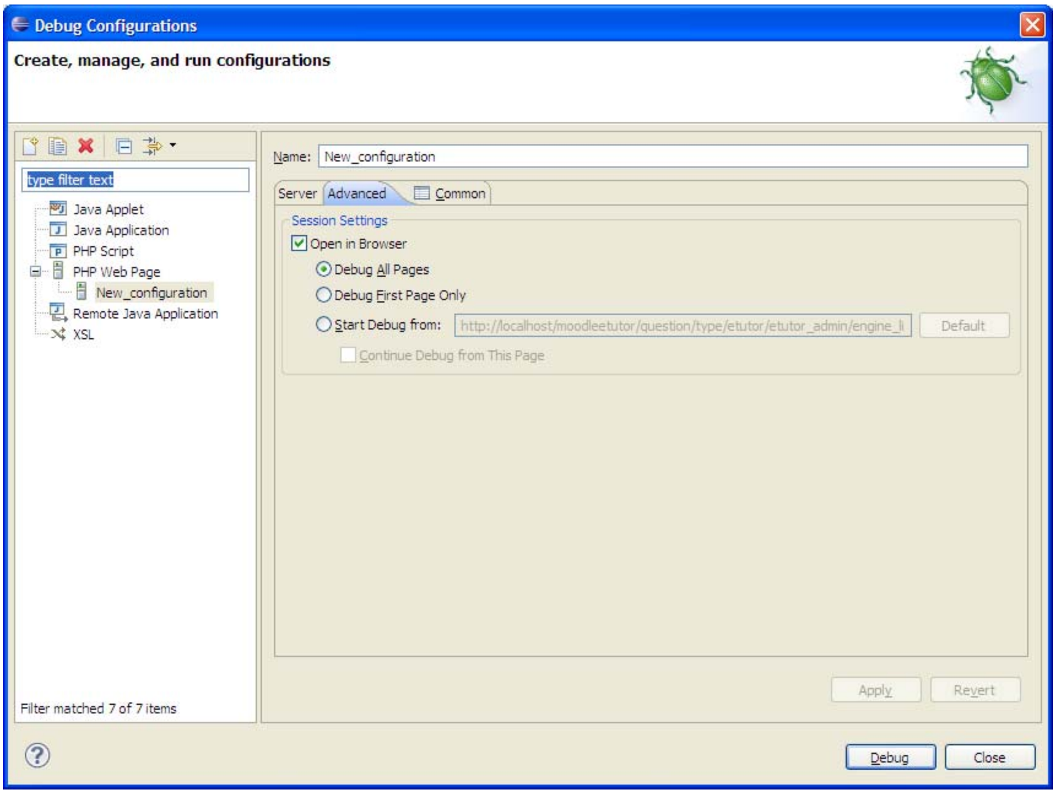


Abbildung 13 - Debugging-Einstellungen 2

2.3 Installation des eTutor-Plugins

An dieser Stelle wird erklärt, wie das im Rahmen der Diplomarbeit entwickelte Plugin installiert wird. Die Installation für eine neue Moodle-Instanz behandelt Kapitel 2.3.1; die Integration des Plugins in eine vorhandene Moodle-Instanz wird in Kapitel 2.3.2 beschrieben.

2.3.1 Installation in neuer Moodle-Instanz

Die Moodle-Installation kann als Gesamtpaket innerhalb von Eclipse aus dem SVN-Repository ausgecheckt werden. Nach Möglichkeit sollte als Zielverzeichnis der htdocs-Ordner von XAMPP gewählt werden. Wenn das Projekt in einem anderen Ordner liegt, muss sein Inhalt in den htdocs-Ordner kopiert werden.

Für die Installation von Moodle sollte zunächst die Datei *config.php* im Hauptverzeichnis gelöscht werden. Anschließend wird beim Aufruf von Moodle im Webbrowser der Installationsassistent gestartet um Moodle einzurichten. Um die vorbereiteten Testdaten einspielen zu können, kann als Tabellenpräfix die Standardeinstellung *mdl_* beibehalten werden.

2.3.2 Installation in vorhandener Moodle-Instanz

Bei einer bereits vorhandenen Moodle-Installation müssen die entwickelten Moodle-Plugins an mehrere Stellen aus der Versionsverwaltung in den Moodle-Server kopiert werden. Der im Folgenden kursiv angegebene Pfad entspricht dabei dem relativen Pfad unter \implementation\moodleetutor in der Versionsverwaltung und dem *htdocs/moodle*-Verzeichnis am Webserver der Moodle-Installation.

- eTutor-Fragentyp - */question/type/etutor*
- eTutor-Administrationsblock - */blocks/etutor_admin*
- Tutorenbewertungsblock - */blocks/dke_tutor*

Damit die neuen Datenbanktabellen angelegt werden und die DKE-Tutor-Rolle eingerichtet wird, muss der cron-job durchlaufen werden. Dieser kann manuell über die URL *http://localhost/moodleetutor/admin/cron.php* angestoßen werden.

2.4 Rollen editieren

Nach der Installation müssen unter dem Menüpunkt *Nutzer/Zugriffsrechte/Rollen verwalten* der Moodle-Administrationsansicht[23] folgende Rolleneinstellungen angepasst werden.

- Änderungen bei der Rolle *Administrator*:
    - „*block/dke_tutor:tutorassignment*“ unterbinden
- Eigene Rolle *DKE-Tutor* erstellen (abgeleitet von der Rolle *Trainer ohne Bearbeitungsrecht*)
- Änderungen bei der Rolle *DKE-Tutor*:
    - „*block/dke_tutor:tutorassignment*“ erlauben

[23] http://localhost/moodleetutor/admin/roles/manage.php

- *„block/dke_tutor:gradestudentsubmission“* erlauben
- *„moodle/course:view“* erlauben, damit sich der Tutor nicht im Kurs inskribieren muss

Eine weitere Änderung betrifft die Rollenzuweisungen. Unter dem Menüpunkt *Rechte zur Rollenzuweisung* sollte in der Zeile *Trainer* die Spalte *DKE-Tutor* angekreuzt werden. Somit können Kursleiter in ihren Kursen die Rolle DKE-Tutor selbst vergeben.

	Administrator	Kursverwalter	Trainer	Trainer ohne Bearbeitungsrecht	Teilnehmer	Gast	Authentifizierter Nutzer	DKE-Tutor
Administrator	☑	☑	☑	☑	☑	☑	☐	☑
Kursverwalter	☐	☐	☑	☑	☑	☑	☐	☐
Trainer	☐	☐	☐	☑	☑	☑	☐	→ ☑
Trainer ohne Bearbeitungsrecht	☐	☐	☐	☐	☐	☐	☐	☐
Teilnehmer	☐	☐	☐	☐	☐	☐	☐	☐
Gast	☐	☐	☐	☐	☐	☐	☐	☐
Authentifizierter Nutzer	☐	☐	☐	☐	☐	☐	☐	☐
DKE-Tutor	☐	☐	☐	☐	☐	☐	☐	☐

Abbildung 14 - Rechte zur Rollenzuweisung

Hinweis	Weitere Rolleneinstellungen werden im Benutzerhandbuch (Anhang A) beschrieben. So können beispielsweise die Eigenschaften der Rolle DKE-Tutor dahingehend angepasst werden, dass Tutoren als Administrator für Diskussionsforen eingesetzt werden können.

2.5 E-Tutor-Integration

Die Testdaten in der Datei questioninsert.sql müssen dahingehend angepasst werden, dass die korrekte URL zum *etutor-dispatcher*–Server angegeben wird:

```
insert into mdl_question_etutor_server (name, url, selected)
values
('etutor local', 'http://localhost:8080/etutor/services/ETutorService',
'y')
```

Codeausschnitt 11 - eTutor-Verbindungsdaten in questioninsert.sql[24] anpassen

Hinweis	Die Verbindungseinstellungen zum eTutor-Dispatcher können auch über einen Benutzerdialog[25] angepasst werden. Diese Einstellungen werden im Administrationshandbuch (Anhang C) näher behandelt.

Die Daten von questioninsert.sql können anschließend in die Moodle-Datenbank eingespielt werden. Damit werden die Expertenmodule und einige Testfragen zum SQL-Modul konfiguriert.

[24] /implementation/Testdaten Moodle/questioninsert.sql
[25] http://localhost/moodleetutor/question/type/etutor/etutor_admin/engine_list.php

Über den Link *E-Tutor-Verbindung* im Block *E-Tutor-Admin* wird eine Liste der E-Tutor-Server angezeigt. Nach Einspielen der Testdaten ist bereits eine Verbindung im System vorhanden. Mit einem Klick auf das Lupensymbol kann man überprüfen, ob damit ein Webservice-Zugriff auf den eTutor-Server möglich ist (siehe dazu auch das Administrationshandbuch (Anhang C)).

Nach der Konfiguration des eTutor-Systems wurde Moodle mit eTutor-spezifischen Erweiterungen installiert. Daraufhin wurden die beiden Systeme integriert und Benutzerrollen konfiguriert. Die Basisinstallation ist somit abgeschlossen. Im Rahmen der Umstellung vom alten eTutor-System muss außerdem eine Migration des Datenbestands vorgenommen werden, welche im Diplomarbeitsdokument[26] beschrieben wird.

Hinweis	Weitere Schritte wie das Einrichten eines Kurses und das Erstellen von Übungsaufgaben werden im Benutzerhandbuch (Anhang A) beschrieben.

[26] /documentation/DA.docx

C. Administrationshandbuch

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeausschnittverzeichnis

1 eTutor

In diesem Dokument werden Anpassungsmöglichkeiten hinsichtlich der Integration der Lernsysteme Moodle und eTutor beschrieben; als Zielgruppe können daher Systemadministratoren genannt werden. Kapitel 1 beschreibt Einstellungen der eTutor-Installation und Kapitel 2 jene der Moodle-Installation. Weitere Informationen zum eTutor-System finden sich in der eTutor-Architekturbeschreibung[27]; Einstellungsmöglichkeiten des Moodle-Kernsystems sind Teil des Moodle-Handbuchs[28].

1.1 Expertenmodul hinzufügen

Bevor ein neues Expertenmodul installiert werden kann, müssen die vorgegebenen Methoden der standardisierten Modulschnittstelle implementiert werden. Kapitel 1.1.1 beschreibt, welche Anpassungen an einem Expertenmodul durchgeführt werden müssen, damit dieses mit dem im Rahmen der Diplomarbeit entwickelten eTutor-Dispatcher kompatibel ist.

1.1.1 Anforderungen an Expertenmodule

Die benötigten Klassen eines Expertenmoduls können aus den Einträgen der Spring-Konfigurationsdatei abgeleitet werden. In dieser XML-Datei wird festgelegt, welche Komponenten voneinander abhängen. Diese Konfiguration außerhalb des Quellcodes wird auch als lose Kopplung bezeichnet.Im Folgenden ist die Konfiguration des SQL-Expertenmoduls angeführt:

```
<bean id="evaluatorList" class="java.util.HashMap">
      <constructor-arg>
            <map>
                  <entry key="sql" value-ref="sqlEvaluator" />
                  ...
            </map>
      </constructor-arg>
</bean>
<bean id="showEditorViewList" class="java.util.HashMap">
      <constructor-arg>
            <map>
                  <entry key="sql" value-ref="sqlShowEditorView" />
                  ...
            </map>
      </constructor-arg>
</bean>
<bean id="printReportViewList" class="java.util.HashMap">
      <constructor-arg>
            <map>
                  <entry key="sql" value-ref="sqlPrintReportView" />
                  ...
            </map>
      </constructor-arg>
</bean>
<bean id="exerciseSettingViewList" class="java.util.HashMap">
      <constructor-arg>
            <map>
                  <entry key="sql" value-ref="sqlExerciseSettingView" />
```

27 /documentation/projektdokumentation/architektur.doc
28 http://docs.moodle.org/

```
                ...
            </map>
        </constructor-arg>
</bean>
<bean id="editorList" class="java.util.HashMap">
        <constructor-arg>
            <map>
                <entry key="sql" value-ref="sqlEditor" />
                ...
            </map>
        </constructor-arg>
</bean>
<bean id="exerciseManagerList" class="java.util.HashMap">
        <constructor-arg>
            <map>
                <entry key="sql" value-ref="sqlExerciseManager" />
                ...
            </map>
        </constructor-arg>
</bean>
```

Codeausschnitt 1 - applicationContext.xml[29]

Die *SQLEvaluator*-Klasse realisiert das Interface *etutor.core.evaluation.Evaluator* und implementiert die automatische Korrektur von Übungsaufgaben, die Erstellung einer Rückmeldung an den Kursteilnehmer und die Vergabe der erreichten Punkte.

```
public interface Evaluator {

        public Analysis analyze(int exerciseID, int userID, Map
passedAttributes, Map passedParameters) throws Exception;

        public Grading grade(Analysis analysis, int maxPoints, Map
passedAttributes, Map passedParameters) throws Exception;

        public Report report(Analysis analysis, Grading grading, Map
passedAttributes, Map passedParameters, Locale locale) throws Exception;
}
```

Codeausschnitt 2 - Evaluator-Interface

Im *Evaluator*-Interface ist die Methode *analyze* vorgegeben, die eine zum Übungsbeispiel hinterlegte Musterlösung mit der Abgabe des Studenten vergleicht; die daraus resultierenden Erkenntnisse werden in einem Analyse-Objekt (*etutor.core.evaluation.Analysis*) abgelegt. Aus dieser Analyse berechnet die *grade*-Methode eine Punktzahl (*etutor.core.evaluation.Grading*) und die *report*-Methode erstellt daraus eine individuelle Rückmeldung an den Studenten (*etutor.core.evaluation.Report*).

```
public interface Editor {

        public HashMap<String, Object> initPerformTask(int exerciseId) throws
Exception;

        public HashMap<String, Object> preparePerformTask(HashMap<String,
```

[29] /implementation/etutor_dispatcher/WebContent/WEB-INF/conf/applicationContext.xml

```
Object> passedAttributes, HashMap<String, Object> passedParameters,
Resource [] resources) throws Exception;

      public HashMap<String, Object> prepareAuthorTask(HashMap<String,
Object> passedAttributes, HashMap<String, Object> passedParameters,
Resource [] resources, Serializable exerciseInfo) throws Exception;
}
```

Codeausschnitt 3 - Editor-Interface

Die *initPerformTask*-Methode gibt eine Liste von Initialisierungsparametern zur Übungsausarbeitung zurück, die in der Benutzersession abgelegt werden. Beim SQL-Expertenmodul sind diese Parameter jedoch nicht vorgesehen, weshalb die Implementierung dieser Methode bei der Klasse *SQLEditor* den Wert *null* zurückgibt. *PreparePerformTask* verarbeitet die Eingaben des Benutzers und legt diese in der Benutzersession ab. So liefert diese Methode im *SQLEditor* die vom Studenten eingegebene Abfrage als String zurück. Ihr Pedant im Rahmen der Beispielerstellung ist die Methode *prepareAuthorTask*.

```
public interface ModuleExerciseManager {

      public int createExercise(Serializable exercise,
                  Map attributes, Map parameters) throws Exception;

      public boolean modifyExercise(int exerciseId, Serializable exercise,
                  Map attributes, Map parameters) throws Exception;

      public boolean deleteExercise(int exerciseID) throws Exception;

      public Serializable fetchExercise(int exerciseID) throws Exception;

      public Serializable fetchExerciseInfo() throws Exception;

      public String generateHtml(Serializable exercise, Locale locale)
        throws Exception;
}
```

Codeausschnitt 4 - ModuleExerciseManager-Interface

Die Methode *createExercise* der Klasse *etutor.modules.sql.SQLExerciseManager* erstellt eine neue Übung und liefert als Rückgabewert deren Identifier. Diese ID ergibt sich durch Inkrementieren der in der Datenbank maximal vorhandenen ID. Mittels *modifyExercise* werden vorhandene Übungsbeispiele angepasst; die *deleteExercise*–Methode dient zum Löschen von Übungsbeispielen.

Die beiden *fetch*-Methoden des *ModuleExerciseManager*-Interface unterscheiden sich dahingehend, dass die *fetchExercise*–Methode Beispieldaten einer vorhandenen Übung zurückgibt, während *fetchExerciseInfo* ein Beispielobjekt zum Erstellen einer neuen Übung liefert. Die Implementierung der Methode *generateHtml* ist optional und hängt von der Fähigkeit des jeweiligen Expertenmoduls ab, den Angabetext eines Übungsbeispiels automatisch generieren zu können. Da beispielsweise basierend auf dem SQL-Statement der Musterlösung nicht automatisch eine aussagekräftige textuelle

Beispielangabe generiert werden kann, ist diese Methode im *SQLExerciseManager* nicht implementiert.

Jedes Expertenmodul beinhaltet folgende Benutzerdialoge:

- **ShowEditorView** – Dieses Formular ist je nach Beispieltyp unterschiedlich gestaltet. Beispielsweise enthält es beim SQL-Modul ein Textfeld zum Absetzen der Datenbankabfrage.
- **PrintReportView** – Im Report-Dialog wird eine Rückmeldung zur Studentenabgabe angezeigt; diese wird automatisch vom Expertenmodul erstellt.
- **ExerciseSettingView** – Mit diesem Dialog können neue Übungen erstellt und existierende Übungsbeispiele geändert werden. Die Implementierung dieses Dialogs ist optional; alternativ kann der beispieltypübergreifende *etutor.core.ui.GenericExerciseSettingView* eingesetzt werden.

Alle Dialoge liefern ein *etutor.core.ui.Response*-Objekt, das neben dem anzuzeigenden HTML-Code und einem CSS-Stylesheet eine Liste benötigter Dateien enthalten kann. Dadurch ist es möglich, in Benutzerdialogen Bilder und JavaScript-Funktionen einzubinden.

1.1.2 Expertenmodul einbinden

Für die Neuentwicklung eines Expertenmoduls empfiehlt es sich, ein vorhandenes Expertenmodul (beispielsweise etutor_sql) zu kopieren und anzupassen. Der in diesem Kapitel beschriebene Ablauf zeigt, wie man neue Expertenmodule in das eTutor-System, den eTutor-Dispatcher, einbindet. Die Beschreibung erfolgt exemplarisch an der Installation des SQL-Expertenmoduls.

1.1.2.1 Benutzerdialog installieren

Jedes Expertenmodul ist als eigenständiges Java-Projekt realisiert. Die kompilierten Class-Dateien werden im eTutor-Dispatcher-Projekt referenziert. Im Gegensatz dazu müssen die restlichen Expertenmodul-spezifischen Dateien (beispielsweise Dialogbeschreibungen in Form von Velocity-Templates, Bilder und JavaScript-Dateien) direkt in das eTutor-Dispatcher-Projekt kopiert werden. Beim SQL-Expertenmodul sind die Dialogdateien beispielsweise in folgenden Orten im eTutor-Dispatcher-Projekt abgelegt.

- /etutor_dispatcher/scr/resources/sql/exercisesettingview
  - create_exercise.js
  - sqlExerciseSettingView.vm
- /etutor_dispatcher/scr/resources/sql/printreportview
  - report.css
  - report.js
  - sqlPrintReportView.vm
- /etutor_dispatcher/scr/resources/sql/showeditorview

- taskEditor.css
- taskEditor.js
- sqlShowEditorView.vm

1.1.2.2 Spring-Konfiguration anpassen

Die Klassen des SQL-Expertenmoduls werden in der Spring-Konfigurationsdatei eingetragen:

```
<bean id="sqlEvaluator" class="etutor.modules.sql.SQLEvaluator" />

<bean id="sqlPrintReportView"
class="etutor.modules.sql.ui.SQLPrintReportView" >
  <property name="velocityEngine">
    <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
    <list>
      <value>/resources/sql/printreportview/report.js</value>
    </list>
  </property>
</bean>

<bean id="sqlShowEditorView"
class="etutor.modules.sql.ui.SQLShowEditorView" >
  <property name="velocityEngine">
    <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
    <list>
      <value>/resources/sql/showeditorview/taskEditor.js</value>
    </list>
  </property>
  <property name="cssFilenames">
    <list>
      <value>/resources/sql/showeditorview/taskEditor.css</value>
    </list>
  </property>
</bean>

<bean id="sqlExerciseSettingView"
class="etutor.modules.sql.ui.SQLExerciseSettingView" >
  <property name="velocityEngine">
    <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
    <list>
      <value>/resources/sql/exercisesettingview/create_exercise.js</value>
    </list>
  </property>
</bean>

<bean id="sqlEditor" class="etutor.modules.sql.ui.SQLEditor" />

<bean id="sqlExerciseManager"
class="etutor.modules.sql.SQLExerciseManager" />

<bean id="sqlVersion" class="java.lang.String">
  <constructor-arg value="3" />
</bean>
```

```
<bean id="sqlImplementedFeedbacklevel" class="java.lang.String">
  <constructor-arg value="3" />
</bean>
```

Codeausschnitt 5 - SQL-Expertenmodul-relevanter Ausschnitt der applicationContext.xml

Der Parameter *sqlVersion* wird bei jeder Änderung einer Dialog-Datei inkrementiert; er wird für das Zwischenspeichern der Dateien in Moodle benötigt. Bei einer Erhöhung des *Version*-Parameters werden alle in Moodle zwischengespeicherten Dateien verworfen und erneut vom eTutor übertragen (siehe Beschreibung auf Seite 53).

1.1.2.3 BuildPath

In den Projekteinstellungen des Expertenmoduls können unter dem Unterpunkt *Java Build Path* im Karteireiter *Libraries* die benötigten *Velocity*- und *Spring*-Libraries (Bibliotheken) eingebunden werden. Wie derartige Bibliotheken erstellt werden können, ist im Installationshandbuch (Anhang B) beschrieben.

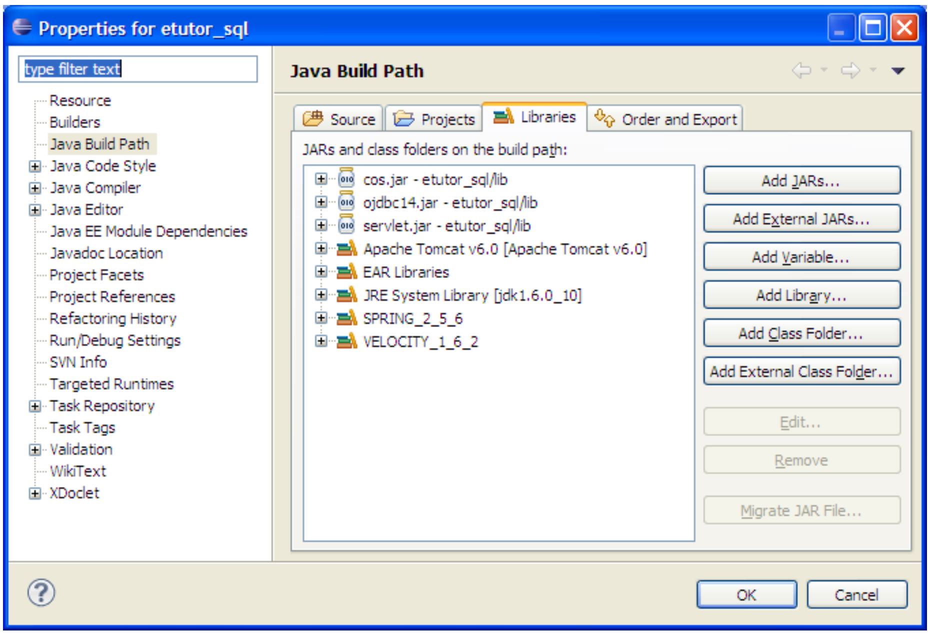


Abbildung 1 - Einbindung der vom Expertenmodul benötigten Bibliotheken

Da Expertenmodule auf Klassen und Interfaces des eTutor-Dispatchers zugreifen, muss sich dieses Projekt ebenfalls im Klassenpfad des Expertenmodul-Projekts befinden, was im Karteireiter *Projects* konfiguriert werden kann.

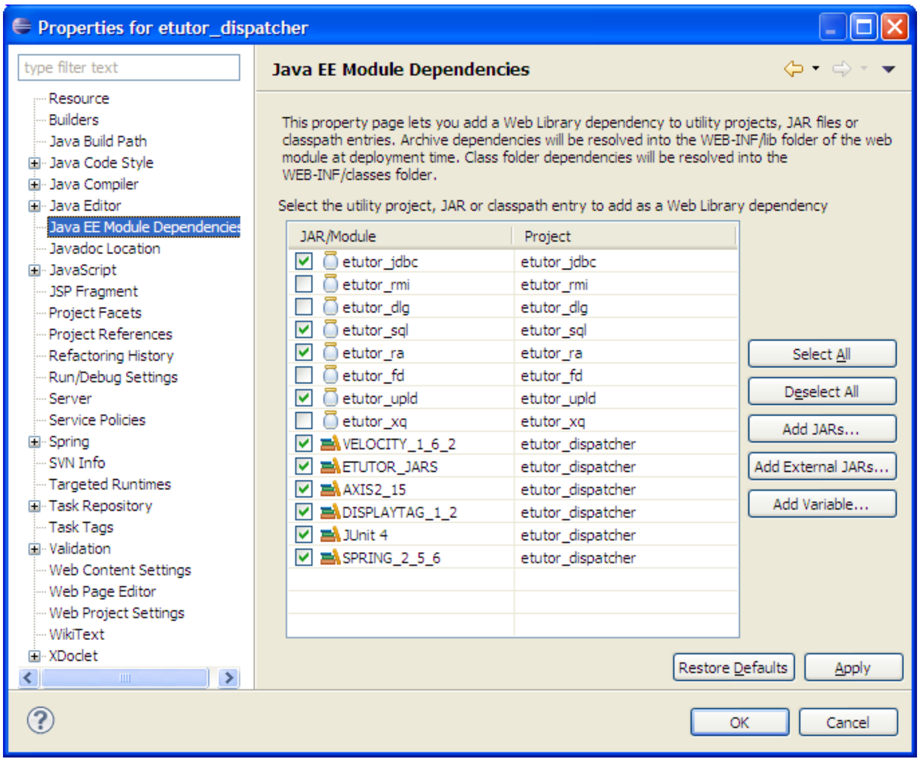


Abbildung 2 - Deployment von Expertenmodulen und Libraries festlegen

Damit referenzierte Projekte und Bibliotheken beim Erstellen der Webanwendung mit auf den Webserver kopiert werden, müssen diese in den eTutor-Dispatcher-Projekteinstellungen (siehe Abbildung 2) unter dem Menüpunkt *Java EE Module Dependencies* selektiert werden.

1.2 Ausgabetext ändern

Soll ein Ausgabetext geändert werden, beispielsweise eine fehlerhafte Übersetzung, muss die Konfigurationsdatei der jeweiligen Sprache (beispielsweise messages_de.properties für deutschsprachige Texte) angepasst werden.

…
`decomposereporter.first=First`
`decomposereporter.second=Second`
`decomposereporter.third=Third`
…
	…
	`decomposereporter.first=Erste`
	`decomposereporter.second=Zweite`
	`decomposereporter.third=Dritte`
	…

(a) **messages_en.properties**[30]	**(b) messages_de.properties**[31]

Codeausschnitt 6 - Ausgabetext anpassen

In Codeausschnitt 6 ist ersichtlich, dass jeder Sprachdatei-Eintrag in einer eigenen Zeile steht und aus einem eindeutigen Schlüssel (zB *decomposereporter.first*) und einem sprachabhängigen Text (*First* oder *Erste*) besteht.

[30] /implementation/etutor_dispatcher/src/messages_en.properties
[31] /implementation/etutor_dispatcher/src/messages_de.properties

1.3 Sprache hinzufügen

Die Auslagerung der Benutzertexte in Textdateien hat den Vorteil, dass Sprachen ohne Modifikation des Quelltexts hinzugefügt oder entfernt werden können.

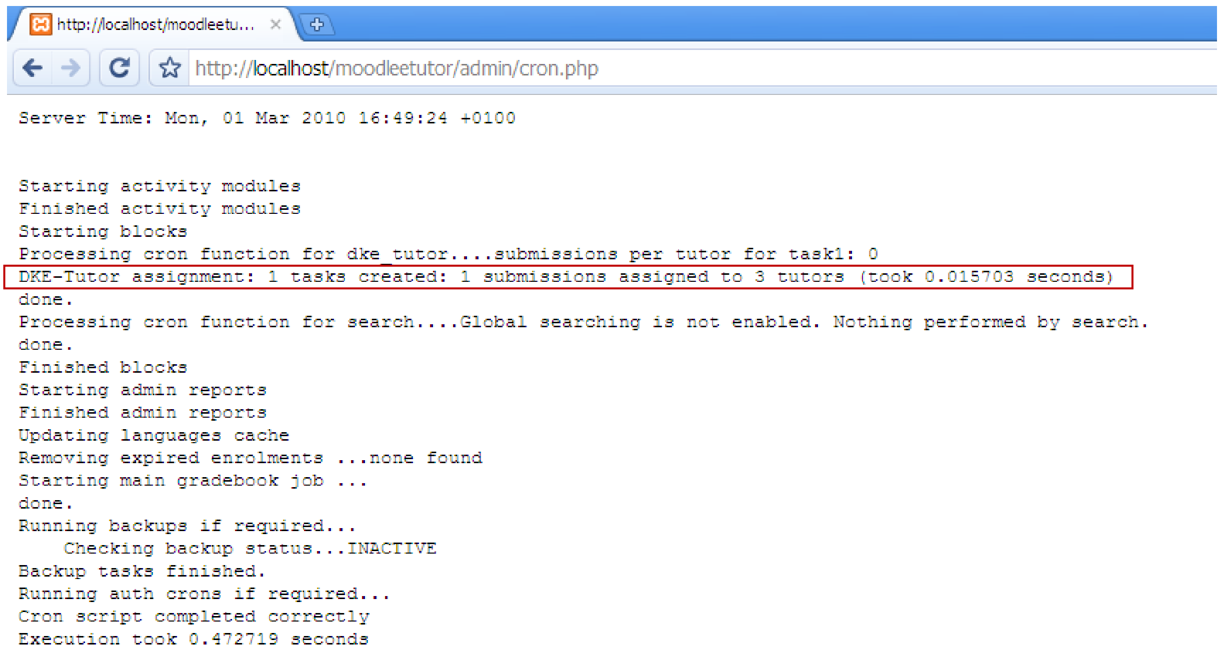
```
<bean
class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
  <property name="targetClass" value="etutor.core.utils.WsUtils"/>
  <property name="targetMethod" value="setLocaleLanguageCode"/>
  <property name="arguments">
    <map>
      <entry key="de_utf8" >
        <util:constant static-field="java.util.Locale.GERMAN"/>
      </entry>
      <entry key="en_utf8" >
        <util:constant static-field="java.util.Locale.ENGLISH"/>
      </entry>
    </map>
  </property>
</bean>
```

Codeausschnitt 7 - konfigurierte Sprachen in applicationContext.xml

Dem Konstruktor der Klasse *WsUtils* wird in der Spring-Konfigurationsdatei eine Liste der konfigurierten Sprachen übergeben. Der *Key*-Wert gibt das Sprachkürzel im Moodle-System an.

Zudem muss für jede in der Spring-Konfigurationsdatei angeführte Sprache eine Properties-Datei mit den lokalisierten Texten existieren. Die Sprachdateien sind im Ordner */implementation/etutor_dispatcher/src/* abgelegt; der Dateiname hat die Form *messages_SPRACHCODE.properties* (beispielsweise messages_de.properties).

1.4 Übungsaufgaben an Tutoren zuteilen




```
Server Time: Mon, 01 Mar 2010 16:49:24 +0100


Starting activity modules
Finished activity modules
Starting blocks
Processing cron function for dke_tutor....submissions per tutor for task1: 0
DKE-Tutor assignment: 1 tasks created: 1 submissions assigned to 3 tutors (took 0.015703 seconds)
done.
Processing cron function for search....Global searching is not enabled. Nothing performed by search.
done.
Finished blocks
Starting admin reports
Finished admin reports
Updating languages cache
Removing expired enrolments ...none found
Starting main gradebook job ...
done.
Running backups if required...
    Checking backup status...INACTIVE
Backup tasks finished.
Running auth crons if required...
Cron script completed correctly
Execution took 0.472719 seconds
```

Abbildung 3 - Zuteilung von Übungsausarbeitungen an Tutoren mittels Cron-Job

Bei manuell korrigierten Übungsbeispielen werden zur Übungsausarbeitung Tutoren zugeteilt. Diese Tutoren-Zuordnung wird nach Übungs-Abgabeschluss mittels Cron-Job vorgenommen. Benutzer mit *Administratoren*-Rolle können diesen Prozess über die URL *http://localhost/moodleetutor/admin/cron.php* auch manuell anstoßen.

1.5 SOAP-Kommunikation anpassen

An dieser Stelle werden Anpassungsmöglichkeiten der Webservice-Schnittstelle des eTutor-Dispatchers beschrieben. Eine Architekturbeschreibung des eTutor-Webservice findet sich im Diplomarbeitsdokument[32].

Damit Änderungen der SOAP-Funktionsschnittstellen des eTutor-Dispatchers sofort wirksam werden, sollte am Moodle-Server die Einstellung *soap.wsdl_cache_enabled* der PHP-Konfigurationsdatei[33] auf den Wert *0* gesetzt werden. In der Standardeinstellung (Wert *1*) wird die Web Service Description Language (WSDL)-Konfiguration zwischengespeichert und somit würde bei einer Änderung der Schnittstellenbeschreibung der Zugriff auf eine veraltete Version der WSDL-Datei stattfinden.

1.5.1 Webservice-Funktionen hinzufügen

Soll das Webservice um weitere Funktionalität erweitert werden, müssen die neuen Methoden sowohl in der Webservice-Beschreibungsdatei ETutor.wsdl[34] als auch in der Klasse etutor.core.ws.service.MyETutorService[35] der eTutor-Dispatcher-Webanwendung eingetragen werden.

1.5.2 Generierung des AXIS2-Codes

Im Folgenden wird beschrieben, wie aus einer WSDL-Datei der Quelltext eines Axis 2-Webservers erstellt wird. Diese Vorgehensweise entspricht dem Contract First-Ansatz. Im Gegensatz dazu würde bei Verfolgung eines Code First-Ansatzes ausgehend von der Servicefunktionalität eine WSDL-Datei erzeugt werden.

```
bin\wsdl2java.bat
 -uri D:\DA\svn\implementation\webservice_definition\ETutor.wsdl
 -o C:\GeneratedServer
 -ss
 -or
 -ns2p
  http://etutor.dke.uni-linz.ac.at/etutor/service/=etutor.core.ws.service,
  http://etutor.dke.uni-linz.ac.at/etutor/types/=etutor.core.ws.types
 –sd
```

Codeausschnitt 8 - Contract First-Erstellung des eTutor-Webservice mittels wsdl2java.bat

[32] /documentation/DA.docx
[33] /XAMPP-Installationsverzeichnis/php/php.ini
[34] /implementation/webservice_definition/ETutor.wsdl
[35] /implementation/etutor_dispatcher/src/etutor/core/ws/service/MyETutorService.java

Der in Codeausschnitt 8 dargestellte Aufruf des Skripts *wsdl2java* erzeugt aus der eTutor-Schnittstellenbeschreibung (*ETutor.wsdl*) den Webservice-Code des eTutor-Dispatchers. Diese Vorgehensweise entspricht dem Contract First-Ansatz. Vor der Codegenerierung sollte sichergestellt sein, dass das Zielverzeichnis leer ist. Die dafür verwendeten Parameter sind in Abbildung 4 beschrieben.

-uri	Dateisystem-Pfad der WSDL-Datei
-o	Zielspeicherort der generierten Java-Klassen und Konfigurationsdateien
-ss	gibt an, ob serverseitiger Webservice-Code generiert werden soll
-or	gibt an, ob existierende Dateien im Zielverzeichnis überschrieben werden sollen
-ns2p	Mit dieser Option werden XML-Namespaces die angegebenen Packages der generierten Java-Klassen zugewiesen.
-sd	zusätzlich zum Quelltext des Webservice wird die Konfigurationsdatei services.xml erstellt (dieser Parameter kann nur in Kombination mit -ss verwendet werden)

Abbildung 4 - Parameterbeschreibung der Datei wsdl2java.bat

1.5.3 Anpassung des generierten Codes

An dieser Stelle muss die Webservice-Konfigurationsdatei[36] angepasst werden, damit der Webservice-Listener mit der Service-Methode des eTutor-Dispatchers (*eTutorService*) kommunizieren kann. Dazu werden, wie in Codeausschnitt 9 dargestellt, die Einträge *ServiceObjectSupplier* und *SpringBeanName* ergänzt.

[36] /GeneratedServer/resources/services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file was auto-generated from WSDL -->
<!-- by the Apache Axis2 version: 1.5  Built on : Apr 30, 2009 (06:07:24
EDT) -->
<serviceGroup>
  <service name="ETutorService">
    <messageReceivers>
      <messageReceiver mep="http://www.w3.org/ns/wsdl/in-out"
class="etutor.core.ws.service.ETutorServiceMessageReceiverInOut"/>
    </messageReceivers>
    <parameter
name="ServiceClass">etutor.core.ws.service.MyETutorService</parameter>
    ...
    <parameter name="ServiceObjectSupplier" locked="false">
      org.apache.axis2.extensions.spring.receivers.
        SpringServletContextObjectSupplier
    </parameter>
    <parameter name="SpringBeanName"
      locked="false">etutorService
    </parameter>
    ...
  </service>
</serviceGroup>
</xml>
```

Codeausschnitt 9 - services.xml

1.5.4 Archivdatei erstellen

An dieser Stelle sind alle Vorbereitungen getroffen, dass der Webservice-Quelltext mit dem Werkzeug *Ant*[37] kompiliert und in ein aar-Archiv konvertiert werden kann. Dazu gibt man über die Kommandozeile im Quellcode-Zielverzeichnis (hier: *C:\GeneratedClient*) den Befehl *ant* ein, wodurch die in der Make-Datei[38] angegebenen Befehle ausgeführt werden.

```
ant
```

Codeausschnitt 10 - Compilierung des Webservices

Ergebnis der ant-Skript-Ausführung ist unter anderem eine Webservice-Archivdatei[39]. Dabei handelt es sich um eine komprimierte Dateisammlung, die mit einem Datenkompressionsprogramm (wie beispielsweise WinRar[40]) geöffnet werden kann.

Mit Ausnahme der im Folgenden aufgelisteten Dateien können aus dem Webservice-Archiv alle Ordner und Dateien entfernt werden:

- /etutor/core/ws/service/
    - ETutorServiceMessageReceiverInOut.class
    - ETutorServiceSkeleton.class
- /META-INF/

[37] http://ant.apache.org/
[38] C:/GeneratedClient/build.xml
[39] C:/GeneratedClient/build/lib/ETutorService.aar
[40] http://www.winrar.de/

- ETutor.xsd
- ETutorService.wsdl
- MANIFEST.MF
- services.xml

Danach muss die kompilierte Java-Klasse *MyETutorService.class*[41] in das Unterverzeichnis */etutor/core/ws/service/* des Archivs kopiert werden.

1.5.5 Installation in Webanwendung

Das Webservice-Archiv *eTutorService.aar* wird zur Installation in den Unterordner *WebContent\WEB-INF\services* der eTutor-Dispatcher-Webanwendung[42] kopiert. Außerdem müssen aus dem Build-Ordner folgende Klassen in die Webanwendung kopiert werden:

- Alle Klassen im Package etutor.core.ws.types[43]
- etutor.core.ws.service.ETutorFaultMessage.java
- etutor.core.ws.service.ETutorServiceSkeleton.java

1.6 SOAP-Debugging

In der Axis 2-Distribution ist mit dem SOAP-Monitor[44] ein Werkzeug enthalten, mit dem die Kommunikation zwischen Moodle und dem eTutor-Dispatcher betrachtet werden kann. Der SOAP-Monitor listet dazu chronologisch alle Nachrichten und deren Antworten im XML-Format auf.

[41] C:/GeneratedClient/\build\classes\etutor\core\ws\service\MyETutorService.class
[42] \implementation\etutor_dispatcher\WebContent\WEB-INF\services
[43] C:/GeneratedClient/build/classes/etutor/core/ws/types/
[44] http://localhost:8080/etutor/SOAPMonitor

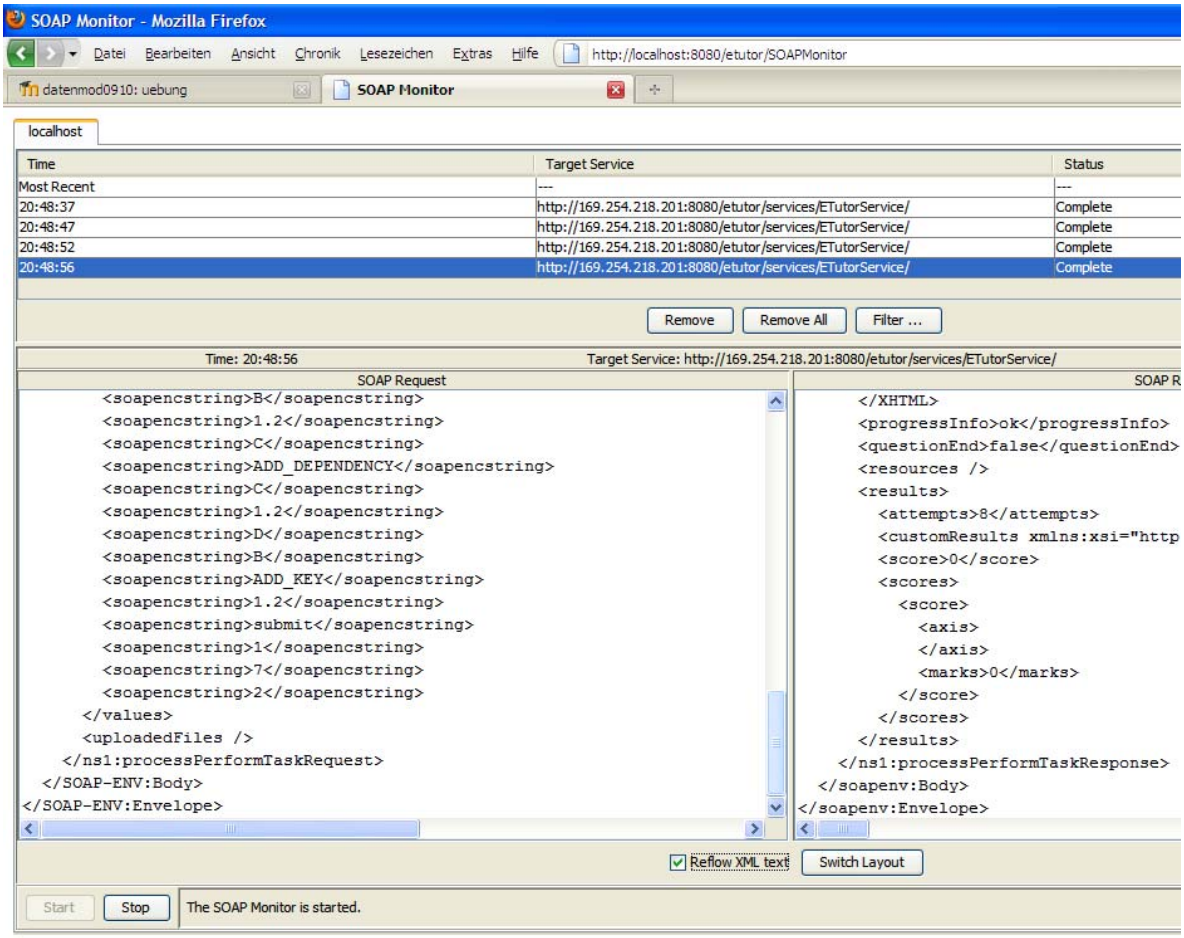


Abbildung 5 - SOAP-Kommunikation beobachten

Treten Fehlermeldungen in Bezug auf die eTutor-Integration auf, geben folgende Webservice-Fehlercodes Hinweise auf die Fehler-Art:

Fehlercode	Bedeutung
1	Fehler beim Laden der Benutzersession
2	Ein benötigter Parameter der Webservice-Anfrage wurde nicht gefunden
11	Im eTutor-Dispatcher ist eine Null-Pointer-Exception aufgetreten
12	Ein für die Webservice-Antwortnachricht benötigter Parameter wurde nicht gesetzt.
13	Zur angegebenen ID wurde keine Aufgabe gefunden.
99	Sonstiger Fehler

Abbildung 6 - Erklärung von Webservice-Fehlercodes

1.7 Sessionverwaltung administrieren

```
<bean id="sessionManager"
class="etutor.core.ws.session.impl.SessionManagerImpl">
  <!-- timeout -->
  <constructor-arg index="0" value="14400000" />
  <!-- check-interval -->
  <constructor-arg index="1" value="600000" />
</bean>
```

Codeausschnitt 11 - Einstellungen des Session-Managers in applicationContext.xml

In der Konfigurationsdatei Spring-applicationContext.xml[45] des eTutor-Dispatchers wird im ersten Parameter des *SessionManager*-Konstruktors festgelegt, nach welcher Zeitspanne Sessions verfallen. Der zweite Parameter bestimmt das Intervall, in dem das Ablaufdatum der Sessions überprüft werden soll. Beide Werte werden in Millisekunden angegeben.

[45] /implementation/etutor_dispatcher/WebContent/WEB-INF/conf/applicationContext.xml

2 Moodle

In diesem Kapitel wird die Konfiguration der im Zuge der Diplomarbeit entwickelten Moodle-Erweiterungen beschrieben. Administrationsbeschreibungen der Moodle-Basisfunktionalität befinden sich im Moodle-Handbuch[46].

2.1 Verbindung zum eTutor-Dispatcher anpassen

Moodle greift mittels Webservice-Schnittstelle auf den eTutor zu. Aus diesem Grund muss in Moodle die Verbindung zum eTutor-Dispatcher konfiguriert werden.

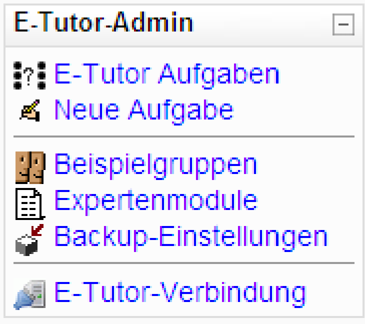


Abbildung 7 - Kursblock E-Tutor-Admin

Der Einstellungsdialog ist über den Kursblock E-Tutor-Admin erreichbar. Zunächst wird eine Liste bisher konfigurierter Verbindungen angezeigt, wobei eine davon als *aktiv* gekennzeichnet ist. Eine Verbindungseinstellung besteht aus dem Namen und einer Serveradresse. Zudem wird im Einstellungsdialog festgelegt, ob die jeweilige Verbindung zur Kommunikation mit dem eTutor verwendet werden soll.

Abbildung 8 - Verbindung zum eTutor-Dispatcher konfigurieren

Nach der Erstellung einer eTutor-Verbindung scheint diese in der Übersichtsliste auf.

[46] http://docs.moodle.org/en/Main_Page

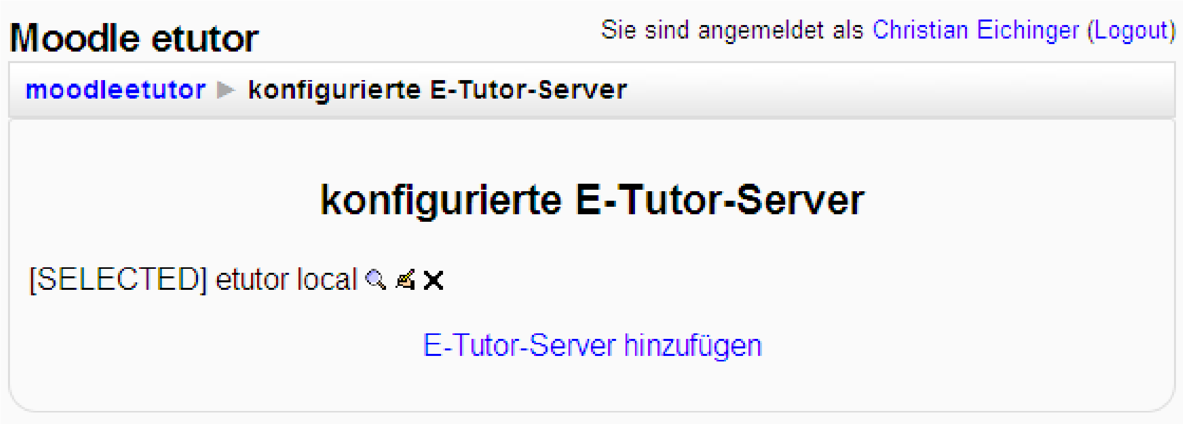


Abbildung 9 - Auflistung von eTutor-Verbindungen

Das bei jeder aufgelisteten Verbindung angegebene Lupensymbol dient dazu, die Konnektivität zum eTutor-Dispatcher zu überprüfen. Ist der Verbindungsaufbau erfolgreich, werden Systeminformationen wie die aktuelle Speicherauslastung des eTutor-Dispatchers angezeigt.

2.2 Expertenmodul konfigurieren

Unter dem Menüpunkt *Expertenmodule* des *eTutor-Dispatcher*-Blocks können Beispieltypen administriert werden. Zunächst werden die aktuell konfigurierten Expertenmodule aufgelistet, wobei diese aus der Ansicht heraus gelöscht oder editiert werden können. Die Verknüpfung *Beispieltyp hinzufügen* führt zu einem Dialog zum Konfigurieren eines neuen Beispieltyps. Zunächst muss ein neuer Beispieltyp im eTutor-System eingebunden werden (siehe Kapitel 1.1.2), bevor im Moodle-Einstellungsdialog mittels des *Kurzbezeichnung*-Attributs auf den in eTutor konfigurierten Beispieltyp verwiesen werden kann. Die Einstellung *Expertenmodulversion* wird für den Zwischenspeicher von Expertenmoduldateien auf Moodle-Seite benötigt. Wenn dieses Versionsattribut nicht mit der auf eTutor-Dispatcher-Seite eingestellten Version übereinstimmt, werden alle zwischengespeicherten Dateien neu übertragen (siehe für eine umfassendere Erklärung des Zwischenspeichers das Diplomarbeitsdokument[47]).

Im Zuge einer Expertenmodul-Erstellung muss angegeben werden, ob beim jeweiligen Beispieltyp eine Funktion zur automatischen Generierung der textuellen Angabe implementiert ist. So besitzt das Expertenmodul bei Schlüsselbestimmungs-Übungen alle benötigten Informationen, um die textuelle Beschreibung der Aufgabenstellung über die Relationsattribute zu bestimmen (siehe zum Anlegen eines Übungsbeispiels das Benutzerhandbuch (Anhang A)).

[47] /documentation/DA.docx

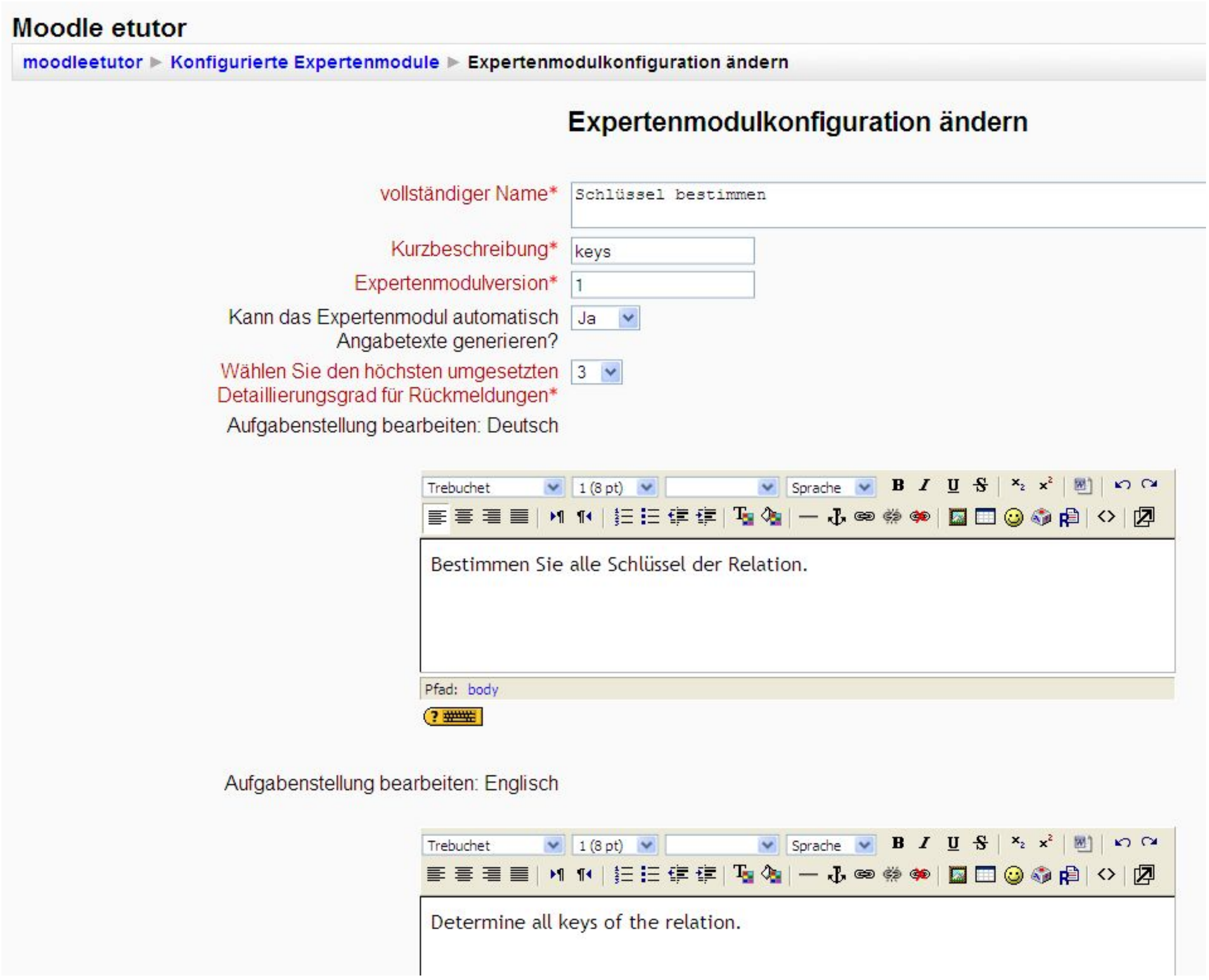


Abbildung 10 - Dialog zum Hinzufügen eines Beispieltyps

2.3 Sprache hinzufügen

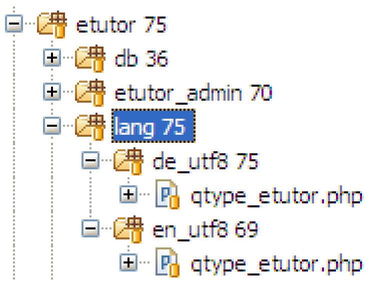


Abbildung 11 - Die Sprache von qtype_etutor.php ist über den Ordnernamen ersichtlich

Moodle ermöglicht es, Ausgabetexte aus dem Code herauszulösen, und in Konfigurationsdateien mehrsprachig bereitzustellen. Dazu befindet sich im Unterordner *lang*[48] des jeweiligen Erweiterungsmoduls für alle unterstützten Sprachen eine Datei mit den jeweils lokalisierten Texten.

[48] \implementation\moodleetutor\question\type\etutor\lang,
\implementation\moodleetutor\blocks\etutor_admin\lang,
\implementation\moodleetutor\blocks\dke_tutor\lang

2.4 Text ändern

Um einen Ausgabetext zu editieren, muss die entsprechende Sprachdatei geöffnet werden (zur Beschreibung des Ablageorts der Sprachdateien siehe Kapitel 2.3). Alle Texte sind als Zeichenketten in einem Array abgelegt. Dementsprechend muss beim Modifizieren der Ausgabetexte auf die korrekte PHP-Syntax geachtet werden.

`$string['searchquestions'] = 'Fragen suchen';` `$string['norestriction'] = 'keine Einschränkung';`	`$string['searchquestions'] = 'Search questions';` `$string['norestriction'] = 'no restriction';`
(a) Deutsch	**(b) Englisch**

Codeausschnitt 12 - Ausgabetexte editieren